Master Thesis Applied Mathematics

Mixed Integer (Non-) Linear Programming Formulations of Graph Neural Networks

T. H. J. N. Mc Donald
4467051
November 2022

TU Delft

Mixed Integer (Non-) Linear Programming Formulations of Graph Neural Networks

by

T. H. J. N. Mc Donald

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday November 11, 2022 at 13:00.

Student number:	4467051	
Project duration:	Feb 1, 2022 – Nov 1, 2022	
Thesis committee:	A/Prof. A. M. Schweidtmann,	TU Delft, supervisor
	A/Prof. N. Yorke-Smith,	TU Delft, supervisor
	Prof. K. I. Aardal,	TU Delft, formal supervisor

TU Delft

Abstract

Recently, ReLU neural networks have been modelled as constraints in mixed integer linear programming (MILP) enabling surrogate-based optimisation in various domains as well as efficient solution of machine learning verification problems. However, previous works have been limited to multilayer perceptrons (MLPs). The Graph Convolutional Neural Network (GCN) model and the GraphSAGE model can learn from non-euclidean data structures efficiently. We propose a bilinear formulation for ReLU GCNs and a MILP formulation for ReLU GraphSAGE models. We compare our formulations to a Genetic Algorithm (GA) by comparing solution times and optimality gaps while modelling a dataset of boiling points of different molecules. Our method guarantees to solve optimisation problems with trained GNNs embedded to global optimality. Between our two formulations the GraphSAGE neural network achieves similar model accuracy, and achieves faster solving times when embedded as a surrogate model in an MILP problem. Finally, we present a computer aided molecular design (CAMD) case study where the formulations of the trained GNNs are used to find molecules with optimal boiling points.

Acknowledgements

As I sit here on a Thursday night finishing up the final comments of my thesis I would like to take the time to thank those who have aided me in finalising this thesis over the last 8 months. However, I also want to thank those who have helped me over the last years of my higher education. Not only does the finalisation of this thesis represent the completion of my Master Applied Mathematics in Delft, but it also represents the closing of the chapter for me as a student.

First of all, I would like to thank Artur and Neil for their combined efforts in supporting me throughout this process. Thank you for giving me inspiration for the topic, which eventually evolved into the research that we can read in this thesis. I would like to thank both for the great conversations, the inspiration for research directions, and tips while being stuck. There were many instances throughout this process where a 30 minute conversation with either of you helped me resolve problems which over the days prior, I wasn't able to solve on my own.

I would also like to thank Calvin Tsay for taking a look at my thesis and giving feedback, although not formally involved in the thesis at all. Your curiosity, and the ideas that followed from your interest in the subject helped me a lot in setting up the experiments of the thesis. Another academic who I would like to thank is Bjarne Grimstad. Your research has inspired me and provided an example to hold onto for this thesis.

I would like thank my friends who I have met living in Lochem, Utrecht and Amsterdam. First of all, for the great conversations, unconditional friendship and great days and nights we spent together. But also, our friendship has always reminded me of my passions outside of mathematics and gave me the opportunity to be a more well rounded individual. With regards to my thesis, I would like to thank you for taking the time to try to understand what this long piece of text was about. Throughout the many talks we had, I improved my own understanding of the research topic and was sometimes struck with inspiration for how to continue the thesis. Finally, I would like to thank my roommates in particular, who have made writing this thesis 10 times more enjoyable through morning coffees, blowing off steam in the weekend, wholesome evening meals and general great support throughout the process.

I would also like to thank my brother for the many runs and beers we shared since I moved to Amsterdam. I want to thank you for your good advice filled with solutions and pragmatism and also the great friendship that has developed since ever since we live so close together here in Amsterdam.

Finally, I want to thank my parents for their unconditional support up until this point in my life. You have always supported me in irregardless of where my interests lay. You have given me the opportunity to follow my passions all the while providing me with a stable foundation to fall back on. You have made me the person I am today and for that I am eternally grateful.

Tom McDonald

3 November, 2022

Amsterdam

Contents

1 Introduction

The modelling and designing of molecules has long been an interest to researchers and has a wide variety of application domains. The domains where these methods can be applied range anywhere from fuel design, resulting in molecules with decreased emissions, to designing molecules for drug discovery, possibly saving human lives. In the past, these methods mostly relied on human expertise and experimentation. Recently, Computer Aided Molecular Design (CAMD) has been introduced. In molecular design, CAMD methods are used to pre-screen a large number of molecules, such that the most promising candidates can be investigated for further testing, saving time and resources of researchers.

An early method used for CAMD was the Quantitative Structure Property Relationship (QSPR) method. With QSPR methods, chemical descriptors are analysed of a group of molecules and then used to predict chemical properties. There are a wide variety of descriptors which are used in QSPR methods. On a large scale, examples of chemical descriptors include group counting where atom groups are analysed [1, 2]. Descriptors can also go to a micro scale in case of quantum-chemical descriptors for instance, where examples include dipole and H-bonding parameters [3]. For a large group of molecules, a variety of chemical predictors are recorded, and then used in regressions. The resulting regression model can be used to predict particular chemical properties of molecules. Often in molecular design, it is interesting to have a molecule where a property is maximised or minimised. There are different methods to achieve this. For instance, using mixed integer linear programming (MILP) formulations of these QSPR regressions and optimising them [4]. The goal is to optimise an approximation of a particular property and find a corresponding input molecule which corresponds to the maximal value. A drawback of QSPR methods is that they are heavily dependant on the knowledge of researchers to select which chemical descriptors are important. Machine learning (ML) models circumvent this problem.

The advent of machine learning models and the increased availability of large data sets, resulted in an increased interest in using ML for prediction tasks. There are a large variety of machine learning methods, one of which are neural networks. Supervised neural networks learn non-linear relationships from a large labelled data set, by trying to match input data to output data [5]. There are different neural network architectures, one of them is a feed forward multilayer perceptron (MLP). An MLP emulates the structure of the brain, where the similarity stems from the fact that an MLP also has neurons with an activation threshold [5]. This threshold is encoded in what is known as an activation function, where the neurons get activated after reaching a predetermined input threshold. The neurons are organised in layers, including an input layer, hidden layers and output layer. The outputs of every consecutive layer, are weighted and used as the input for a neuron in a following layer [5]. This translates into mathematical terms as a composition of consecutive affine layers and activation layers. This architecture allows the MLP to find non-linear relationships in data [5].

There have been applications of MLPs in CAMD. Most instances use MLPs in the QSPR methods, where linear regression is replaced by a MLP to perform a regression [4]. The chemical descriptors of the molecules are used as an input and there is a single neuron as the output layer. The advantage being that feature selection is not important, as the MLP model selects, through learning, which properties are important. However, we were not able to find instances where the molecule alone is used as an input. This is because it is difficult to capture spatial information of the molecule with an MLP since an MLP has a vector input. Other neural network architectures are developed for non-euclidean input data types. An example of one of these networks is the graph neural network (GNN).

GNNs are neural networks which learn using a mathematical graph as input for learning. Accompanying the spatial graph information, every node in the graph also has an associated feature vector, storing information about that particular node. The information of a node gets passed through an MLP for every node in the network. However, the information that gets passed through the MLP for a node is not only the feature vector of that node, but also the feature vectors of the neighbouring nodes in the graph [6]. This allows GNNs to take spatial information into consideration when learning-non euclidean data. There are two GNN architectures which we consider in this thesis. The first being the Graph Convolutional Neural Network by Kipf and Welling [7]. This neural network is one of the earliest graph neural networks and is used often in GNN applications. The second is the GraphSAGE network by Hamilton et al. [8], which learns properties of large graph data by sampling the neighbourhood of nodes instead of using information of all neighbouring nodes.

Molecules can also be represented as graphs. Every atom in a molecule is represented by a node, and the properties of this molecule are stored in the feature vectors associated with the atom-representing nodes. There have been multiple studies where GNNs have been used to predict properties of molecules (see [9] for an overview). To use these methods in CAMD, just as with the previously mentioned QSPR methods, one wants to optimise the modelled properties and see which molecule corresponds to this optimised value. Rittig et al. [10] have done exactly that, using Bayesian optimisation and a genetic algorithm to optimise the trained GNNs. These methods are not deterministic optimisation methods. This means that found solution might be the local maximum of the trained GNN and not the global maximum. In many cases, it is favourable to know with certainty that the found solution is the global optimum.

Recently, MILP formulations have been introduced of Rectified Linear Unit (ReLU) MLPs. ReLU MLPs, are MLPs where the activation function is a piece-wise linear function called the ReLU function. Due to its piece wise linear nature the activation function is able to be expressed with linear programming constraints using big-M constraints. The other functions in an MLP are affine and thus the whole network can be linearised. The class of MILP problems are able to be solved to global optimality using commercial solvers. This new research area has been applied to a wide variety of topics like MLP verification [11–14], compression of MLPs [15, 16] and using MLPs as surrogate models in linear programming problems [17–20].

To this day, we have not found an MILP formulation of a trained GNN in the literature. We believe this can be interesting addition to the literature as it can be used for similar applications as MLPs, like verification of GNNs, compression of GNNs and using GNNs as surrogate models in optimisation problems. We also believe that MILP formulations for GNNs can be used in CAMD, where properties of molecules can be modelled using GNNs and then optimised using MILP formulations of these trained GNNs. For these reasons we believe there are benefits to be had to find an MILP formulation of GNNs and therefore, we proposed the following research questions at the start of thesis:

1. Can we formulate Graph Convolutional Networks (GCN) as Mixed Integer (Non-) Linear Problems?
2. Can we achieve similar model accuracy with the GraphSAGE GNN of which the architecture extends more naturally to MILP formulations resulting in decreased solving times?

Answering these research questions resulted in a few different research contributions which where directly used to answer the research question, but also resulted in adjacent contributions which were a by product of the research. The found contributions are the following:

- We propose a mixed integer non-linear programming formulation of the frequently used Graph Convolutional Network model by Kipf and Welling [7].
- We propose a mixed integer linear programming formulation of the GraphSAGE model by Hamilton et al. [8] without sampling the neighbourhood of nodes and with the add pooling function as aggregator. We propose this method as it allows us to formulate a completely linear formulation of the GNN as opposed to the MINLP formulation of the GCN.
- We propose a new MILP formulation of the molecular input space based on a structure conducive to and inspired by graph neural networks. The input space is constrained using a combination of feature vectors and an adjacency matrix.
- We propose a genetic algorithm implementation to optimise GNNs which does not depend on latent space architecture as proposed by Rittig et al. [21]. The proposed method uses a string representation of the symmetric adjacency matrix of the molecules and of the feature vectors of the molecules such that single point crossovers and string mutations can be applied.
- We present a case study where we optimise the boiling points of molecules modelled with the GraphSAGE and GCN models. The trained MI(N)LP formulations of the trained GNNs were optimised and the results were inspected.

The thesis has the following outline. Chapter 2 contains a literature review. The chapter lays out the relevant literature on QSPR methods, graph neural networks, the optimisation of feed forward MLPs, and finally concludes with a section on CAMD. Chapter 3 explains all the theory which is required to understand the rest of the thesis, including sections on linear programming, MLPs, an introduction to the architecture of the GCN [7] and GraphSAGE [8] model, and finally an MILP formulation of MLPs. Thereafter, Chapter 4, introduces the novel theory introduced in this thesis. First we introduce a MINLP formulation of the GCN model. Thereafter, an MILP formulation of the GraphSAGE model is introduced. This is followed by a section on how to constrain an input space such that these formulations can be used to look for molecule-like structures in optimisation problems. There is a section on bound tightening techniques for GNNs, and the chapter concludes with a section on a genetic algorithm which can be used to optimise GNNs. Chapter 5, contains a section on the experimental setup, after which the results of these experiments and the case study are presented. The thesis concludes with Chapter 6, where the research questions are answered, the results are discussed, and finally there is a section placing the research in context and further research possibilities are laid out.

1.1 Abbreviations and Notations

1.1.1 Abbreviations

Abbreviation	Meaning
CAMD	Computer Aided Molecular Design
ChebNet	Chebyshev spectral Graph Neural Network
CNN	Convolutional Neural Nets
Conv-GNN	Convolutional Graph Neural Network
Dist-GNN	Distinct Graph Neural Network
FBBT	Feasibility Based Bound Tightening
GA	Genetic Algorithm
GCN	Graph Convolutional Network
GNN	Graph Neural Network
ILP	Integer Linear Programming
MILP	Mixed Integer Linear Program
MINLP	Mixed Integer Non Linear Program
ML	Machine Learning
MLP	Multilayer Perceptron
MSE	Mean Square Error
OBBT	optimisation Based Bound Tightening
QSPR	Quantitative Structure Property Relationship
Rec-GNN	Recurrent Graph Neural Network
ReLU	Rectified Linear Unit

Table 1: A list of all abbreviations in this thesis in alphabetical order

1.1.2 Notations

Notation	Meaning
Graphs	
G	graph
V	set of vertices in graph G
E	set of edges in graph G
v	node in set V
e_{vw}	edge in set E from node v to w
A	adjacency matrix of a graph
d_{max}	maximum degree of the graph
n	cardinality of set V
	number of nodes in a graph G
Linear Programming	
z^*_{LP}	optimal value of the LP
z^*_{MILP}	optimal value of the MILP
M	big-M value
Genetic Algorithm	
Y	initial population
f	fitness function
C	chromosome
O	offspring
T	atom sequence
OT	offspring atom sequence
P_m	mutation probability in chromosome
$P_m a$	mutation probability in atom sequence
S	random degree sequence
MILP of an MLP	
K_{MLP}	number of layers in an MLP
n_k	number of neurons in a layer
x^k_j	positive component of neuron j in layer k
$\boldsymbol{x}^k$	vector containing all values of a neuron in layer k
s^k_j	negative component of neuron j in layer k
z^k_j	binary activation variable of neuron j in layer k
$\sigma(z)$	activation function
W^k	weight matrix in layer k
b^k_j	bias value for neuron j in layer k
u^k_j	upper bound of neuron j in layer k
l^k_j	upper bound of neuron j in layer k

MINLP of GCN	
N	number of nodes in a graph
F	number of features in the initial feature vectors
X	all feature vectors for the input of the GCN
A_{ij}	adjacency matrix entry from i to j
I_N	identity matrix of length N
$\tilde{A}$	$A + I_N$, adjacency matrix including self loops
$\tilde{D}_{ii}$, d_i^+	$\sum_j \tilde{A}_{ij}$, degree of node i including self loops
W^k	weight matrix of GCN in layer k
H_{ij}^k	positive component for node i, neuron/feature j, layer k
$\boldsymbol{H}_l^k$	positive component of all neurons/features of node i in layer k
S_{ij}^k	positive component for node i, neuron/feature j, layer k
Z_{ij}^k	activation variable for node i, neuron/feature j, layer k
U_{ij}^k	upper bound for node i, neuron/feature j, layer k
L_{ij}^k	lower bound for node i, neuron/feature j, layer k
$\boldsymbol{b}_{il}^k$	support variable. $\boldsymbol{H}_l^k$ if $\tilde{A}_{il} = 1$, 0 otherwise
p_{il}	support variable indexing degree of node i and node l
c_{il}	support variable indexing degree of node i and node l
s_{il}	variable capturing the value of $\left(\sqrt{d_i^+ d_l^+}\right)^{-1}$
$\hat{s}_{il}$	support variable. s_{il} if $\tilde{A}_{il} = 1$, 0 otherwise
MILP of GraphSAGE	(addition to the GCN symbols)
$\hat{W}^k$	root weight matrix of GraphSAGE in layer k
$\bar{W}^k$	other weight matrix of GraphSAGE in layer k

2 Literature Review

This chapter is an overview of the literature adjacent to the contents of this thesis. It shows which research has already been conducted and where lay the gaps which we are looking to fill with this thesis. This section comprises of the literature on classical chemical property modelling using QSPR methods, followed by relevant papers on neural networks and its applications in chemical property modelling. Thereafter, we discuss the recent advancements in the optimisation of neural networks using mixed integer linear programming formulations of neural networks. The final section focuses on computer aided molecular design.

2.1 QSPR Methods

Molecular property estimation is an important aspect in many fields, including but not limited to drug discovery, material design, process design and biology [21]. An effective prediction method for molecular properties on the basis of the molecules' structural attributes has been QSPR analysis. This mathematical modelling technique finds a correlation between the chemical descriptors of a molecule and the property under investigation [22]. Katritzky et al. [22] categorise QSPR research based on five categories of chemical descriptors. These are constitutional, topological, electrostatic, geometrical and quantum-chemical descriptors. Many examples exist in which regressions include constitutional [22, 23], topological [1, 2], electrostatic [24, 25], geometrical [26] and quantum-chemical [2, 3] descriptors or combinations of these descriptors.

Boiling points have been studied extensively using QSPR methods. Different groups of compounds have been studied, using different correlational techniques and a variety of chemical descriptors. For instance, de Lima Ribeiro and Ferreira [2] study the boiling points of polycyclic aromatic hydrocarbons using thermodynamic, electronic, steric and topological descriptors in a regression model. Roubehie Fissa et al. [27] estimate boiling points of hydrocarbons using multiple linear regression and multi-layer perceptron methods, with a wide variety of descriptors, selected through statistical analysis methods. Dai et al. [28] find that topological descriptors based on the equilibrium electro-negativity of an atom and the relative bond length were effective descriptors to model alkanes, unsaturated hydrocarbons and alcohols.

2.2 Neural Networks

Recently, with the increased amount of computing power and availability of big data, machine learning methods have emerged as an effective method for finding accurate non-linear relationships between data and its properties. Deep learning in particular, a research area of ML, has proven to be a universal approximator provided sufficiently many hidden units are available, even for a single layer neural networks [29]. This fact, in combination without the need of expert intuition for property selection due to the capability of deep learning to automatically learn underlying representations from data, make it an interesting tool for chemistry, drug discovery and chemical engineering [30–32].

Graph neural networks, a subclass of neural networks, are better at regression and classification tasks for non-euclidean data sets than feed-forward neural networks. For a comprehensive review see Wu et al. [6]. Molecules belong to the set of non-euclidean data that are modelled better by GNNs. In a review about GNNs in chemistry Wieder et al. [9] categorise GNNs in chemistry into 3 subgroups. These are (1) Recurrent GNNs (Rec-GNN),

(2) Convolutional GNNs (Conv-GNN) and (3) Distinct Graph Neural Network Architectures (Dist-GNN). We will consider the first two subcategories as they are relevant for this thesis.

Rec-GNNs where initially introduced by Gori et al. [33] and also introduced the term graph neural networks. This concept was further explored by Scarselli et al. [34] and Gallicchio and Micheli [35]. A Rec-GNN learns a node representation by iteratively applying the same weight matrix over a graph, until an equilibrium state is reached. The simplest Rec-GNNs have been employed to perform property prediction of graph representations of molecules [34, 36, 37]. More complex Rec-GNNs based approaches exist which use gate based architectures, like GRU and LTSM networks (of which more information is available in Wu et al. [6] and Yu et al. [38]).

Convolution graph-neural-networks (Conv-GNN) are categorised as spectral and spatial based approaches. Spectral based GNNs are based on spectral graph theory and use the spectral decomposition of the graphs, in combination with filters to reduce noise from the graph signals [6]. The initial spectral based approach Spectral Convolutional Neural Network (CNN) [39] assumes the filter to be a set of learnable parameters. Spectral CNN utilises eigendecompositions, which are computationally expensive, and are not adaptable to graphs of different sizes. The Chebyshev spectral CNN (ChebNet) [40] was introduced to solve these problems by approximating the weight filter with Chebychev polynomials. Kipf and Welling [7] introduce the Graph Convolutional Network [7] by taking the first order Chebychev approximation, which alleviates the problem of over fitting [7]. Although it is a spectral method it can also be interpreted as a spatial method since the approximation basically results in an aggregation function of the neighbouring nodes of a node to form a convolutional layer [6].

Spatial Conv-GNN methods are conceptually similar to non-graph based convolutional neural nets (CNN), as "spatial-based graph convolutions convolve the central node's representation with its neighbors' representations to derive the updated representation for the central node" [6]. Micheli [41] introduced these spatial graph Neural Networks. Thereafter, many varieties of spatial Graph Neural networks have been introduced. Basic models include PATCHY-SAN, LGCN and GraphSAGE [8, 42, 43]. All use a combination of convolutional operators, combined with different neighbour selection systems and different aggregators. There is also a set of attention-based spatial approaches which assign different weights for different neighbours to minimise noise [44, 45]. Finally, there are more general frameworks which try to unify multiple models in a single formulation as an abstraction over multiple GNNs [46–48].

Since the input structure of graph neural networks extend so naturally to molecules, in recent years it has been applied to many chemical property prediction tasks. All of the previously mentioned graph structures have been applied to learning chemical properties. This includes basic Rec-GNNs [34, 36] and in the gated variants [49–52]. Conv-GNN are not excluded in its application in chemistry, for spectral Conv-GNNs [53, 54] and basic [55, 56], attention [57] and general [47] spatial conv-GNNs. For a complete overview of molecular property prediction with graph neural networks see [9].

Boiling/melting points have also been modelled using graph neural networks. In an early paper by Kireev [58], molecular matrices are used as input to model boiling points on a set of hydrocarbons. Scarselli et al. [34] introduces a RecGNN approach which avoids the need to make graphs directed and rooted, while still using a recurrent architecture. Results are shown to outperform existing QSPR methods while modelling boiling points for alkanes. Yang et al. [59] use a GCN based model called MegNET [60] in which they model melting points with a mean absolute error of less than 6 K.

2.3 Mixed Integer Linear Programming Formulations of Neural Networks

Recently, exact MILP formulations of Neural Networks (NNs) with ReLU activation functions have appeared such that these formulations could be optimised with linear solvers. These exact formulations emulate the ReLU operator using binary activation variables and big-M formulations. There have been various applications like neural network verification [11–14], counting linear regions in Deep Neural Networks [61] and compression of Deep Neural Networks [15, 16]. NNs can also be used as surrogate models. Surrogate models are a simplified approximation of a complex relationship. MILP formulations of NNs allow NNs to be used as surrogate models in optimisation problems. The advantage being that non-linear relationships modelled by neural networks can be expressed in linear optimisation problems [17–20].

From the literature it is apparent that the bounds used in the big-M constraints have an impact on the solving time of the linear solvers [62]. As a result, multiple research papers have sections dedicated on how to tighten said bounds [11, 12, 14, 17, 63]. The most basic and weakest among those is interval arithmetic, where the input bounds are propagated through the neural network [12, 63, 64]. Other BTTs generate two Linear Programming (LP) problems for each neuron in which the bounds of these neurons are minimised and maximised. Tjeng et al. [12] do this with a relaxation of the binary activation variables, which finds better bounds than interval arithmetic but is computationally more expensive. Fischetti and Jo [11] find even tighter bounds by not relaxing the activation variable for a neuron, but this is even more computationally expensive. Wang et al. [65] suggest a method combining the previously mentioned methods. It is a pre-processing approach which comprises of identifying nodes which would benefit most from applying an computationally expensive bound tightening approach.

2.4 Computer Aided Molecular Design

The field of computer aided molecular design is the process of designing molecules for a certain application, by using computers to search the largely unexplored chemical search space. The previously mentioned QSPR method try to predict chemical properties from chemical structures. Traditional CAMD methods attempt the reverse by optimising MI(N)LP formulations of these QSPR models to find structures related to chemical property values [4]. There are many different research papers of QSPR based CAMD, differentiating themselves in the type of molecular descriptors they use and the accompanying MILP formulation of the molecular search space, also referred to as structural feasibility constraints. For a complete overview see Table 3 in Austin et al. [4].

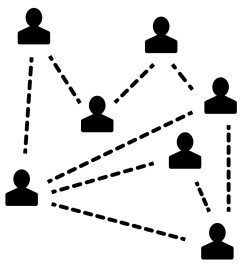
There are different methods to optimise the MI(N)LP formulations used in CAMD. In a survey on CAMD, Austin et al. [4] list two of these methods. First, we consider mathematical optimisation techniques, which are useful in case of many chemical QSPR descriptors, or non-convexities or non-linearities. Examples include using outer-approximation algorithms [66, 67] for MINLP formulations or branch and bound methods [68, 69] for MILP formulations. Otherwise, in case of too large instances or instances where faster solution results are required, heuristic methods are considered. Examples of heuristics used to optimise MI(N)LPs in CAMD are genetic algorithm (GA) [70–74] and Tabu search [75, 76].

Most neural network approaches in CAMD focus on deep generative modelling and using optimisation techniques to find the desired molecules in the search space. Examples of generative molecule models that are employed include Rec-GNNs [77, 78], variational or adversarial autoencoders (VAEs/AAEs) [79–83], generative

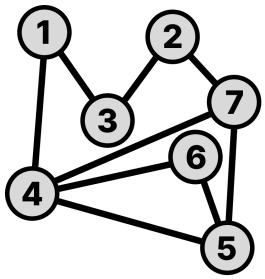
adversarial networks (GANs) [84–86] and reinforcement learning (RL) [87, 88] approaches. Compared to classic CAMD approaches, generative approaches project the molecular learnable encodings onto a continuous latent space, which allows for continuous optimisation techniques to be employed [10].

3 Background

In the following chapter we will go over the background information necessary to understand the presented thesis. This chapter includes a basic introduction to graph theory, linear programming formulations and how to solve these formulations. Thereafter neural networks are introduced and we discuss two graph neural architectures in detail. Finally MILP formulations of neural networks are explained. The informed reader can skip this chapter.



(a) A group of people and their friendship relations



(b) A graph $G = (V, E)$ representing the relation in (a)

$$A = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

(c) An adjacency matrix of the graph G

Figure 1: From friends to graphs

3.1 Graph Theory

We initialise our background theory with a small section on graph theory. A graph $G = (V, E)$ is an abstract representation of objects, and the interactions between them. Every object in the graph is represented by a node $v \in V$. The interactions between two nodes v and w are represented by edges $e_{vw} \in E$ [89]. An example of a network which graphs can represent are friendship networks, where the nodes are people and their friendships are represented by edges. See figure 1(a) for an example.

Nodes and edges in a graph can be represented by an adjacency matrix $A \in \mathbb{R}^{|V|\times|V|}$ and a graph can either be directed or undirected. When a graph is directed, an edge in a graph represents a directed relationship between two nodes v and w [89]. In the adjacency matrix A of a graph, the entry A_{vw} represents an edge e_{vw}. When G is directed, $A_{vw} = 1$ means that there is an edge pointing from node v to node w, and when $A_{vw} = 0$ this edge does not exist. When the graph is undirected, $A_{vw} = 1$ merely represents a relationship between two nodes v and w, and there is no directional aspect to the relation. In this case, the adjacency matrix of graph G is symmetric [89].

A path in a graph is a sequence of nodes such that every two nodes in a sequence are connected by an edge [89]. This means that when there is a path, starting from a node in the sequence of that path, one can reach any other node in that sequence while traversing over a route of active edges. In figure 1(b) an example of a path is the node sequence $P = \{1, 4, 7, 5\}$. In the adjacency matrix in figure 1(c) one can find that $A_{vw} = 1$ for any consecutive node pair v, w in the sequence P.

When a path intersects itself, there is a loop in the sequence. A loop is a path of which the initial and final node in the node sequence of a path are the same node [89], for example the node sequence $R = \{1, 4, 5, 7, 2, 3, 1\}$. When a graph has no loops, the graph is considered to be acyclic, when the opposite is true the graph is classified as cyclic. Since the graph in the example has a loop it is cyclic.

A graph is connected when every node in a graph is reachable from any other node in a graph through a path [89]. An undirected acyclic connected graph is called a tree [89].

Theorem 3.1. *Any connected graph $G = (V, E)$ with n nodes and $n - 1$ edges is a tree.*

Proof. Assume by contradiction that graph $G = (V, E)$ is not a tree. Then there must be a loop in the network. In this case we can remove an edge from the network without disconnecting any part of the graph. We do this repeatedly until no loops are left in the network. The resulting graph has no loops but less than $n - 1$ edges, making it impossible to be connected. Hence G is a tree. □

Corollary 3.1.1. *By definition of a tree, if we have a connected graph $G = (V, E)$ with n nodes and $n-1$ edges the graph is acyclic.*

3.2 Linear Programming

In the next section, linear programming formulations are introduced. Two methods to solve these formulations are explained, namely the branch and bound algorithm and a genetic algorithm. Finally we introduce the big-M method. The following section is adapted from Wolsey [90] unless indicated otherwise.

3.2.1 Definition of an MI(N)LP

In linear programming the goal is to search for an optimal solution in a set of solutions, where the quality of a solution is defined by an objective function. The set of solutions is referred to as the solution space, search space or feasible region. In linear programming, this solution space is defined by a set of linear constraints. These constraints are linear inequalities on the variables $\mathbf{x} \in \mathbb{R}^n$ and these define the solution space $X \in \mathbb{R}^n$. The objective is defined as a linear combination of the variable vector $\mathbf{x}$. In mathematical notation this becomes for a given $\mathbf{b} \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$ and $\mathbf{c} \in \mathbb{R}^n$

$$\max \quad \mathbf{c}^T\mathbf{x} \tag{1a}$$

$$\text{s.t.} \quad A\mathbf{x} \leq \mathbf{b} \tag{1b}$$

$$\mathbf{x} \geq 0. \tag{1c}$$

The feasible region defined by $A\mathbf{x} \leq b$ and $\mathbf{x} \geq 0$ is a polyhedron. In linear optimisation the goal is to find the intersection of the hyperplane $\mathbf{c}^T\mathbf{x}$ and the extreme point of the polyhedron $X_{\text{LP}} \in \mathbb{R}^n$. This is because this intersection is the optimal value of the formulation given in Eqs. (1), in case a feasible and bounded optimum exists. We denote this optimum as $\mathbf{x}^*$. The optimum expressed in terms of the objective function is called the optimal value, $z^*_{LP} = \mathbf{c}^T\mathbf{x}$. It is proven that a linear programming formulation can be solved to optimality in polynomial time using the ellipsoid algorithm. However, when applied, this does not prove to be an efficient algorithm [91]. In practice, the simplex method is employed to solve this problem [92].

There are also optimisation problems where the variables are not expressed as real numbers, but as integer values. Consider a case where students are allotted to rooms for an optimal class schedule. In this instance it is not useful to express students as real valued variables, as 1.37 students cannot be scheduled. For these kind of problems integer linear programming is introduced.

Integer linear programming constrains the variable vector $\mathbf{x}$ to be integer valued. In this case, the standard form optimisation problem from Eqs. (1) become

$$\max \quad \mathbf{c}^T\mathbf{x} \tag{2a}$$

$$\text{s.t.} \quad A\mathbf{x} \leq \mathbf{b} \tag{2b}$$

$$\mathbf{x} \geq 0 \tag{2c}$$

$$\mathbf{x} \in \mathbb{Z}^n. \tag{2d}$$

For intuition, consider Fig. 2(b). The optimality point $\mathbf{x}^*$ is not found at the extreme point of $X_{\text{LP}} \in \mathbb{R}^n$ anymore, but in the intersection of the LP polyhedron and the n dimensional integer values $X_{\text{LP}} \cap \mathbb{Z}^n$. To solve this, one can not utilise the simplex algorithm, or as a matter of fact, any efficient algorithm. Integer linear programming problems are NP-hard in general [93].

A linear programming formulation is a mixed integer linear programming (MILP) formulation when there are both integer $\mathbf{x}$ and real valued $\mathbf{y}$ variables in the formulation. In this instance the standard form in Eqs. (3) is the same, apart from $\mathbf{x}$ begin replaced by $(\mathbf{x}, \mathbf{y})$ and Eq. (3d) being replaced by $(\mathbf{x}, \mathbf{y}) \in \mathbb{Z}^{n_1} \times \mathbb{R}^{n_2}$, where n_1 and n_2 are the amount of integer and continuous variables respectively.

A further generalisation of MILPs are mixed integer non-linear programs. The constraints and objective function of an MINLP can include non-linear functions. The MINLP has the following form:

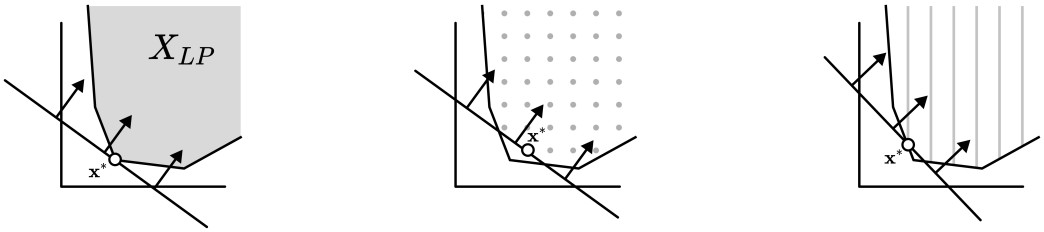
$$\min \quad f^0(x, y) \tag{3a}$$

$$\text{s.t.} \quad f^j(x, y) \leq 0, \qquad j = \{1, \ldots, m\} \tag{3b}$$

$$(\mathbf{x}, \mathbf{y}) \in \mathbb{Z}^{n_1} \times \mathbb{R}^{n_2} \tag{3c}$$

$$(\mathbf{x}, \mathbf{y}) \geq 0 \tag{3d}$$

where at least one of $f^0(x, y)$ and $f^j(x, y)$ is a non-linear function [94]. If all functions $f^0(x, y)$ and $f^j(x, y)$ are convex, the problem is considered convex. If one is non-convex the problem is considered non-convex. Although both the non-convex and convex instances prove to be NP-Hard, the convex instances are considered easier to solve than the non-convex counterpart [94].



(a) Linear Programming (LP): the optimum is found at the intersection of $\mathbf{c}^T\mathbf{x}$ and X

(b) Integer Linear Programming (ILP): the optimum is found in the intersection of $\mathbf{c}^T\mathbf{x}$ and $X_{\text{LP}} \cap \mathbb{Z}^n$

(c) Mixed Integer Linear Programming (MILP): the optimum is found in the intersection of $\mathbf{c}^T\mathbf{x}$ and $X_{\text{LP}} \cap \mathbb{Z}^{n_1} \times \mathbb{R}^{n_2}$

Figure 2: LP vs ILP vs MILP

3.2.2 Solving MI(N)LPs

In this section we will discuss the methods used in this thesis to solve the MI(N)LPs proposed in this thesis. These methods are the branch and bound algorithm and a genetic algorithm.

Branch and bound

The branch and bound algorithm is a deterministic optimisation method. This means that the algorithm will always find a global optimum for MILP problems. The branch and bound algorithm searches the search space using a process called branching, where the MILP is divided into smaller sub problems, which still contain the global optimum.

The branch and bound algorithm initialises a solution by relaxing the integer variables $\mathbf{x}$ in the MILP formulation and finding a solution in polynomial time using the simplex method for example. Due to the relaxation, the feasible region is larger than before. In case of maximisation, we will find an over approximation of the actual optimal value, such that $z^*_{\text{MILP}} \leq z^*_{\text{LP}}$ [93]. Unless the found solution finds integer values for all relaxed integer variables $\mathbf{x}$, there are some $\mathbf{x}$ which will be real valued in the solution. The branch and bound algorithm works by branching on one of these variables x_j to create two sub problems. These two MILP sub problems are the original MILP formulation with a new introduced constraint [93]. Let's say x_j has a fractional value s, then the introduced constraints are $x_j \geq \lceil s \rceil$ and $x_j \leq \lfloor s \rfloor$, creating MILP 1 and MILP 2 respectively. Since only non-integer values are removed by introducing these constraints, the integer solution lies either in MILP 1 or MILP 2. The process is repeated for the newly introduced MILPs creating a search tree [93].

Branching continually will eventually result in finding the integer solution, however this is computationally expensive. There are a few pruning strategies that can be used such that not all branches need to be considered. These are the following:

1. If introducing the new constraints (either $x_j \geq \lceil s \rceil$ or $x_j \leq \lfloor s \rfloor$) reduces the search space of the MILP sub problem to the empty set, then the problem is infeasible. No possible solutions lie in the search space. This branch is considered *pruned by infeasibility*
2. For MILP_i, let LP_i be the relaxed formulation, $(\mathbf{x}^i, \mathbf{y}^i)$ an optimal solution for that formulation and z_i the associated optimal value.
   (a) Consider the case where $\mathbf{x}^i$ is integral, this means we have found an optimal solution for MILP_i, and a valid solution for the original MILP. Branching any further will not improve the found solution for MILP_i. It is considered *pruned by integrality*. Since MILP_i is a constrained version of the original MILP, the search space is smaller and $z_i \leq z^*_{\text{MILP}}$. The optimal value of the relaxed formulation z_i thus also functions as a lower bound.
   (b) In case $\mathbf{x}^i$ is not integral, and z_i is lower than the best known lower bound z_{LB} (assuming it exists, by process of $2(a)$) then no solution will be found which is better than z_{LB}, as branching will only constrain the search space of the newly branched MILPs. We therefore do not need to explore this branch any further and it is *pruned by bound*.

If neither of these pruning strategies apply for the relaxed MILP sub problem, once again the ceiling and floor constraints are introduced on a real valued variable in x_i. Once all the branches of the tree have been considered or pruned, the problem has been solved to global optimality [93].

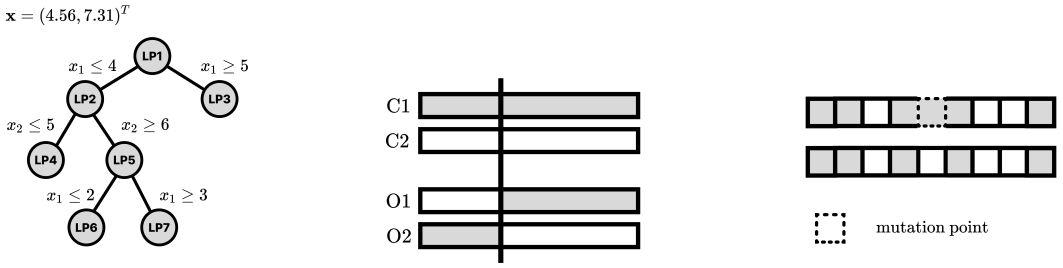
As mentioned above, during the branching, the branch and bound algorithm finds integer feasible solutions

of which the highest objective value z_{LB} acts as a lower bound of the solution of the problem. Also, the solutions of the relaxations of the sub problems act as upper bounds z_{UB}.

The algorithm can be terminated prematurely, for instance when the branch and bound algorithm is terminated after a predefined number of seconds. A benefit of the branch and bound algorithm is that the optimality gap can be calculated. The optimality gap is a relative measure of how far the best found feasible solution is removed from the maximal objective value still possible. The optimality gap is calculated as follows [90]:

$$\delta = \frac{|z_{\text{UB}} - z_{\text{LB}}|}{|z_{\text{LB}}|} \tag{4}$$

Although, when run for enough time, the branch and bound algorithm finds a global optimum, sometimes it is better to consider heuristic methods. These methods find solutions more quickly. However, they do not guarantee a global optimum. One of these heuristics are genetic algorithms.



(a) Example of a search tree of the branch and bound algorithm with an initial solution

(b) An abstract example of a single point crossover. The tail of C1 gets concatenated with the head of C2 and vice versa, to create O1 and O2 respectively.

(c) An abstract representation of the mutation operator. Grey indicates a 1, white a 0. Gene 5 gets flipped from 1 to 0.

Figure 3: Branch and bound and GA explanatory figures

Genetic Algorithm

Genetic algorithms are an evolutionary optimisation methodology. It is evolutionary in the sense that it takes its inspiration from natural selection, and survival of the fittest. There are many variations of genetic algorithms. GAs often take an initial population, represented by bit strings/chromosomes, on which genetic operators are applied in a consecutive manner [95].

In detail, GAs work as follows. An initial random population (Y) of n chromosomes is generated. Their fitness is calculated using some fitness function f. A selection Y_f, also known as elites, of the population Y is chosen based on their fitness value. On this selection, genetic operators will be applied. All elites are matched in pairs. For a pair of chromosomes $C1$ and $C2$ in population Y_f a crossover operator is applied. Thereafter, the two resulting offspring $O1, O2$, have a probability M_p of experiencing a mutation operator. This process is repeated for the pairs in the selected population Y_f. The offspring are used as the population in the next iteration of the algorithm. The algorithm terminates when some kind of stopping criteria is met. In the following paragraphs we will further explore the genetic operators [95].

Solutions for the problem have to be expressed in some kind of encoding (binary, octal, hexadecimal, tree, etc. encoding) [95]. For the purpose of this thesis, we will be restricted to binary encoding.

The first step of a GA is the selection technique. It determines which solutions will participate in the reproduction process. A common selection procedure is the roulette wheel. The roulette wheel selection procedure

randomly selects from the population, where the probability that a particular string i is selected is $p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$. This means that strings with a higher fitness function have a higher probability of being selected to reproduce. Although it is simple to implement, it has problems with the solution prematurely converging to a local minima [95]. In this thesis we also employ a selection procedure called elitism. In this selection procedure all pairs are ranked. The best of these solutions are carried over to the next elite population without any crossover or mutation procedure. To fill the rest of the population, the roulette wheel is used [95].

Cross over procedures generate new offspring of the chromosomes. The most basic of these crossover procedures is the single point crossover. This is where two strings $C1$ and $C2$ of length m are cut in the exact same position t, where $t \leq m$. The crossover position t can be the same or chosen randomly for every iteration. The result after cutting at position t are the strings $[C1_1, \ldots, C1_t], [C1_{t+1}, \ldots, C1_m], [C2_1, \ldots, C2_t]$, and $[C2_{t+1}, \ldots, C2_m]$. The head of $C1$ is concatenated with the tail of $C2$ and vice versa, creating offspring $O1$ ad $O2$ respectively [95]. For clarification please see figure 3(b).

The final iteration in a step of an iteration is the mutation operator. This is where with a probability M_p binary digits of the chromosome get flipped, to create a mutated offspring. The goal is to introduce genetic diversity in the solutions [95]. An figure of the mutation operator can be seen in figure 3(c).

The GA is terminated when a certain stopping criteria is met. An example of this is the number of iterations.

3.2.3 Big-M Formulation

Important in further chapters of the thesis is the big-M constraint. Introducing a big-M constraint can be used as an if statement to check whether a particular variable takes a positive value. To achieve this we introduce a binary activation variable x, which we want to indicate whether a variable y is positive or not. Let M be a known upper bound on y. We impose the following set of constraints

$$y \leq Mx \tag{5}$$

$$x \in \{0, 1\} \tag{6}$$

$$y \geq 0. \tag{7}$$

When $y > 0$, it forces $x = 1$ to be one, and when $y = 0$, x can either be 0 or 1 [93].

An example of the a use case is modelling economic activity, where there is both a fixed and a variable cost. Only once the production y of a product is started, meaning that $y > 0$ we incur the fixed production cost β. The production cost $c = \alpha y + \beta x$, can be modelled in linear programming by inserting the above constraints.

Introducing the above constraints make any LP formulation an MILP formulation due to the introduction of the binary activation variable. The simplex method can not be used to solve this problem anymore. The branch and bound method is often used, meaning that the binary variable is relaxed. The relaxation of this binary variable causes weak bounds for z_{UB} (in case of maximisation) in the branch and bound algorithm as described in Section 3.2.2. It is therefore less likely that the optimal value of a subproblem z_{UB}, is smaller than the best known lower bound. Fewer branches of the search tree will be pruned by bound, and the algorithm will take longer to find a global maximum [93].

3.3 Multilayer Perceptrons

In this section multilayer perceptrons are discussed. First, the inner workings of the model is discussed and thereafter how to train the model.

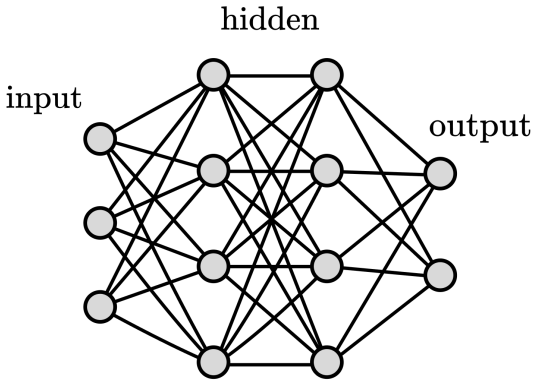


Figure 4: An example of a neural network which can be used for classification with an input layer, 2 hidden layers and an output layer.

3.3.1 Architecture of the Model

A feedfoward multilayer perceptron (MLP) is a mathematical function of which the structure is inspired by neurons in the brain. The network consists of consecutive layers of neurons, which are connected through a directed acyclic network. A neuron in a particular layer gets a weighted signal from the neurons of the previous layer expressed as a real number. Like synapses in the brain, these neurons get activated when the sum of these signals reach a particular threshold. The result of this system is a neural network which has the ability to emulate complex non-linear relationships.

In mathematical terms this translates to a neural network $f(x) : \mathbb{R}^m \mapsto \mathbb{R}^n$ built of multiple layers $k \in \{1, \ldots, K\}$, including the input layer $k = 1$, the hidden layers $k = \{2, \ldots, K-1\}$ and the output layer $k = K$. Each layer contains of n_k neurons. Naturally, the input layer has n_1 neurons and receives the input vector $x_1 \in \mathbb{R}^{n_1}$ of the function. Every layer k has an associated weight matrix $w^k \in \mathbb{R}^{n_k \times n_{k-1}}$ and a bias vector $b^k \in \mathbb{R}^{n_k}$ [5].

The values associated with neurons in consecutive layers $x_k \in \mathbb{R}^{n_k}$ are calculated with a propagation function which is a composition of a set of affine functions and a non-linear activation function. This propagation function takes the inputs from real values of the neurons of the previous layer $x_{k-1} \in \mathbb{R}^{n_{k-1}}$. Thus, for the hidden layers $k = \{2, \ldots, K-1\}$ we have

$$g^k(x^{k-1}) = x^k = \sigma(w^k x^{k-1} + b^k), \tag{8}$$

where $\sigma(y)$ is the activation function. Normally, in the last layer K the activation function is absent and so for x^K we have

$$g^K(x^{K-1}) = x^K = (w^K x^{K-1} + b^K). \tag{9}$$

Completely composed, the neural network $f(x) : \mathbb{R}^{n_1} \mapsto \mathbb{R}^{n_K}$ is defined by

$$f(x^1) = x^K = (g^K \circ g^{K-1} \circ \cdots \circ g^2 \circ g^1)(x_1). \tag{10}$$

[5]

The activation function, indicated by σ in Eq. (8), is a non-linear function, which allows the neural network to find a non-linear relationship between input and output data. There are many different types of activation functions, like the sigmoid, tanh, and ReLU functions. The latter will be the main focus for this this thesis. The ReLU activation function is defined as

$$\sigma(z) = \max\{0, z\}. \tag{11}$$

The function is a piece wise linear function, meaning that it consists of linear segments. Goodfellow et al. [5] recommend using the ReLU function for most feedfoward neural networks. This is due to its simplicity, the fact it preserves many of the properties which make linear models generalise well, and because the function is almost linear making it easy to optimise. We will go into the optimisation of the network in the next sub section.

3.3.2 Training and Testing the Model

The neural network $f(x)$ is unhelpful if the the weight matrix w^k and associated bias b^k of each layer are chosen at random. To emulate a function, the weights and biases of each layer need to be determined through a process called training. There are a few varieties of learning paradigms, commonly separated in supervised learning, unsupervised learning and reinforcement learning [5]. In this thesis, only supervised learning is considered.

Supervised learning uses paired data, where each data point consists of an input vector x, and a desired output y. The goal of the learning task is to tune the the weights and biases to produce the desired output, when the input is propagated through the neural network [5].

In supervised learning there are two types of learning. These are regression and classification [5]. Regression is also known as function approximation. The goal is to predict a numerical output given some input [5]. Linear regression is an example of a supervised regression task.

Classification is when a neural net is tasked with categorising data in specific predetermined groups. An example of a classification learning task can be the categorisation of images of animals. The input vector of the paired data can be an image depicting a cat. This image is flatted from an n by n matrix of pixel values, to an n^2 long vector of pixel values. The data pair also has a desired output. This is a binary vector where each entry represents a different animal. For instance, a vector $y \in \{0, 1\}^3$ where $y = [1, 0, 0]$ represents an image depicting a dog, $y = [0, 1, 0]$ representing a cat, and $y = [0, 0, 1]$ represents a mouse. The goal is to learn which pixel value vectors belong to a desired output y [5].

With a method called back propagation the weights and biases are tuned such that the distance between the desired output and the value determined by the neural net are minimal. The mathematics of this process are not relevant for this thesis but the interested reader can check out section 6.5 in [5]. In the categorisation example, we want to tune the weights and biases such that the distance between the output of the neural net $f(x) : \mathbb{R}^{n^2} \mapsto \mathbb{R}^3$, and the output vector y to be minimal [5].

The distance is determined by the loss functions. There are many different loss functions that can be considered, but in this thesis we consider the mean square error (MSE), which is used for regression tasks. For any given data set with input pairs (x_i, y_i) and trained neural net $f(x_i)$ we calculate the MSE as follows

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2. \tag{12}$$

As the name suggests, it is the mean of the squared difference between the predicted output $f(x_i)$ and the actual output y_i. Another way to interpret the MSE is that it is $\frac{1}{n}$ times the square of the euclidean distance [5].

Hyper-parameters are settings of the machine learning algorithms which are not learnable, which control the behaviour of the learning algorithm. Examples which are relevant for this thesis are the depth, which indicates the number of layers and the width, which indicates the number of neurons per layer. The tuning of hyper-parameters can determine how well the algorithm learns the function it is trying to emulate [5].

If the MLP has considerable depth and width, there is a possibility of over fitting the training data. This occurs when the neural network has learned the training data very well, but when presented with new data, the approximations are less correct. With regards to our loss function, this means that the MSE of the training data could be very small, but when presented with new data, the MSE is considerable again. It has thus only learned the data very well but is not a good approximation of the function which its trying to emulate [5].

To combat this, the available data is often split up in three groups. These groups are the training data, the test data and the validation data. The training data, are the data points which are used to tune the weights and biases of the layers in the MLP, through back propagation. The test data is a separate set of data which is not used during the training phase. This is the data one uses to verify the quality of the model after it is trained [5].

When multiple model configurations are considered (differing in the choice of the hyper-parameters) one can use the validation data to decide which of the model configurations is best. For instance, consider the case where a polynomial is used to perform a regression task, where the polynomial degree is hyper-parameter. The MSE can always be decreased if the degree of the polynomial is increased. One uses the validation set to see which polynomial degree has the lowest loss with unseen data. In the final step, the quality of the model is quantified with the test data [5].

3.4 Graph Neural Networks

MLPs are good at learning data with a vector input. However, there are other architectures which can learn non-euclidean data types better than MLPs. One of those neural network structures are called Graph Neural Networks. These neural networks take a graph structure as the input to regress or classify a data point. There are two GNNs architectures which are used in this thesis. These are, the Graph Convolutional Network model introduced by Kipf and Welling [7] and the GraphSAGE model by Hamilton et al. [8].

3.4.1 General Graph Neural Network Architecture

The data input for graph neural networks are different than for feedfoward neural networks. The data input consists of two components. Every data point consists of the structure of a graph $G = (V, E)$ represented by the adjacency matrix $A \in \mathbb{R}^{N \times N}$, and properties of the graphs. The properties of these graphs are stored in node feature vectors $X \in \mathbb{R}^{N \times F}$, and can sometimes include edge feature vectors, however, we don't consider these in this thesis. Every node $i \in V$ has an accompanying feature vector $X_i \in \mathbb{R}^F$. These feature vectors store information about the node in question. In a supervised setting the data is thus of the form $((X, A), y)$.

As mentioned in Section 2.2, graph convolutional neural networks are divided in spectral and spatial based methods. Spectral based methods are graph neural networks based on graph signal filters. Spatial based

methods are generally GNNs consisting of a function which aggregates neighbourhood information and some sort of propagation function, similar to those found in MLPs. The aggregation function, aggregates the the feature vectors of neighbouring nodes of a node i, which is used as input of an affine function. Thereafter, the affine combination of the aggregated feature vectors is passed through an activation function, similar to the feedfoward neural network architecture. Doing this for every node in the graph constitutes one convolutional layer. After one convolutional layer, every node has a new feature vector.

Stacking multiple convolutional layers consecutively allows a node i to not only process node feature vector information of its neighbouring nodes $\mathcal{N}(i)$, but also of the neighbours $\mathcal{N}(s)$ of these neighbours $s \in \mathcal{N}(i)$. This works as follows. In the first convolutional layer. for every node i, all neighbourhood information is aggregated. In the next layer this happens again, however, all neighbours of node i have already processed the information of their respective neighbours. This means i also internalises the information of all neighbours removed with a 2 length path. After k convolutions, node i processes information from all nodes k-length paths removed.

In the following sections we will discuss the graph aggregation functions of two GNNs as they define the architectures of the GNNs considered in this thesis.

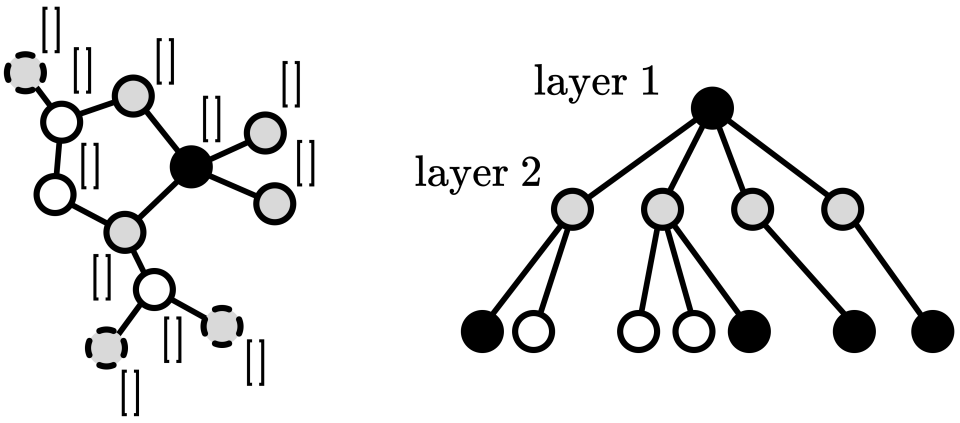


Figure 5: A graph (left) with a feature vector on every node. Neighbourhoods of a node with multiple convolutional layers in a spatial GNN (right)

3.4.2 Graph Convolutional Neural Network (GCN)

The first GNN we consider is the Graph Convolutional Neural Network (GCN) [7]. It is one of the earlier papers which can be considered as a spatial GNN method. The GCN has its roots in spectral graph GNNs as it is a first order Chebychev approximation of the ChebNet [40] architecture, which is a spectral based method. However, this first order approximation is basically a spatial based method.

Kipf and Welling [7] introduce the k-th convolutional layer in the GCN can be expressed as follows:

$$H^{(k+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(k)}W^{(k)}) \tag{13}$$

Here, $\tilde{A} = A + I_N$ is the adjacency matrix of the undirected graph $\mathcal{G}$ with added self-connections. I_N is the identity matrix, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ and $W^{(k)}$ is a layer-specific trainable weight matrix. $\sigma(\cdot)$ denotes an activation function, such as the $\mathrm{ReLU}(\cdot) = \max(0, \cdot)$. $H^{(k)} \in \mathbb{R}^{N \times n_k}$ is the matrix of activations in the k^{th} layer; $H^{(0)} = X$, where X is a matrix of node feature vectors X_i belonging to node i in the graph.

The formula to find feature j for a node i in layer $k+1$ shows the spatial nature of the GCN network:

$$H_{ij}^{(k+1)} = \sigma\left(\left(W_j^{(k)}\right)^T \sum_{l \in N^+(i)} \frac{1}{\sqrt{d^+(i)}\sqrt{d^+(l)}} \left(H_l^{(k)}\right)\right) \tag{14}$$

Here, $\mathcal{N}^+(i)$ is the neighbourhood of i including i itself, and $d^+(i)$ is the degree of node i. The aggregation function is a normalised sum of all the feature vectors of $l \in \mathcal{N}^+(i)$ in layer k. Thereafter, just as in MLP models, an affine combination is taken of the aggregated feature vectors and passed through an activation function σ.

3.4.3 GraphSAGE Network

The GraphSAGE network is also a spatial convolutional neural network. The GraphSAGE network was developed to learn large graph networks. Its input is merely one large graph $G = (V, E)$ on which it performs the learning task. When trained, the GraphSAGE network can classify nodes, without having seen all nodes of the network. This means that it can generalise to unseen nodes in the network.

GraphSAGE also uses an aggregation scheme, for instance the `mean`, `max`, `ltsm` or `add` aggregation scheme. However, for node i, GraphSAGE does not aggregate over all feature vector of its neighbours $\mathcal{N}(i)$, but over a randomised subset of the neighbourhood. This allows it to learn large graphs. The aggregated subset of the neighbourhood vectors gets concatenated with the vector of the root node i. The concatenated vectors are then multiplied with a learned weight matrix $W^k \in \mathbb{R}^{(n_{k+1} \times 2n_k)}$ which consecutively passes through an activation function σ. Finally, the vector gets normalised. As usual, all previously described steps are performed for all nodes $i \in V$.

For this thesis we are interested in the GraphSAGE network as it is linearisable, when specific choices for the hyper-parameters are made. We set the sampling to select all neighbours with a probability of 1. This can be interpreted such that we don't have a sampling function. The chosen activation function is the ReLU function. We choose the aggregate scheme to be `add`, which means that we add all feature vectors of the neighbouring nodes. The propagation function becomes:

$$H_i^{(k+1)} = \sigma\left(H_i^{(k)} \cdot W_1^{(k)} + \sum_{l \in \mathcal{N}(i)} H_l^{(k)} \cdot W_2^{(k)}\right) \tag{15}$$

The matrices $W_1^{(k)}, W_2^{(k)} \in \mathbb{R}^{(n_{k+1} \times n_k)}$ are a split representation of the matrix $W^k \in \mathbb{R}^{(n_{k+1} \times 2n_k)}$, introduced for legibility. Using the add function is also more natural when predicting the boiling points for chemical compounds, which we will discuss in the next section.

3.4.4 GNNs in Chemical Property Prediction

In this thesis we will use GNNs for chemical property prediction. Boiling point prediction is a regression learning task with a single valued output. The GNNs described in the previous sections learn to predict property vectors of nodes. To combat this, some authors combine the convolutional phase with a so called readout phase. After k consecutive convolutional layers, every node has a feature vector. These are combined into a single vector using an aggregation method/pooling function. Examples of these pooling functions are the `max`, `mean` or `sum` operators. The resulting vector is called a fingerprint of the graph G [96]. This vector then gets passed through multiple MLP layers which map to a singular output. In this way we can perform a regression task and learn the parameters using the MSE loss function as described in Section 3.3.

The pooling function that is used in this thesis is the `sum` pooling operator. This is because the 'the nature of the contribution of atoms and bonds [] is expected to be additive' [96]. Meaning that the bigger the molecules are, the more likely they are to have a higher temperature. Using the sum pooling operator naturally makes it such that molecules with more atoms have larger valued vector fingerprints.

3.5 Mixed Integer Linear Programming Formulations of Multilayer Perceptrons

As stated in Section 3.3, training a neural network results in a function which emulates a complex non-linear function. It does so by an architecture of consecutive layers alternating between a set of affine functions and a non-linear activation function. The following section describes how to linearise, for each hidden layer $k \in \{1, \dots, K-1\}$, the following MLP layer:

$$x^k = \sigma(W^k x^{k-1} + b^k) \tag{16}$$

where $\sigma(y) = \max\{0, y\}$ is the ReLU function, $x^k \in \mathbb{R}^{n_k}$ is the output of layer k, W^k and b^k are respectively the found weights and bias of layer k.

There are different methods to linearise Eq. (16). This thesis considers the linearisation by Fischetti and Jo [11]. The output of the affine equations are decoupled in a positive part $x \geq 0$ and negative part $s \geq 0$, resulting in the linear equation

$$w^T y + b = x - s. \tag{17}$$

By doing this, the ReLU function can be emulated by forcing either x or s to be zero. This can be achieved with the introduction of a binary activation variable z and big-M activation constraints. The following logic needs to apply:

$$\begin{cases} z = 0 & \text{then } s \leq 0 \\ z = 1 & \text{then } x \leq 0 \\ z \in \{0, 1\} \end{cases} \tag{18}$$

The linear inequalities that force this logic are big-M constraints. These inequalities are of the type $x \leq M^+(z)$ and $s \leq M^-(1-z)$. The parameters M^+ and M^- are an upperbound and lowerbound on the possible values of x and s respectively. If the left hand side of Eq. (17) is positive, x is forced to positive too. As a result z must be 1 due to its binary property, consecutively forcing $s = 0$. When the left hand side of Eq. (17) is negative the same logic applies, forcing $z = 0$.

It is assumed that bounds can be found such that $l \leq w^T y + b \leq u$. For every neuron j layer k of any neural network where the ReLU function is applied the following set of constraints are introduced:

$$x_j^k \leq u_j^k z_j^k \tag{19a}$$

$$s_j^k \leq -l_j^k (1 - z_j^k) \tag{19b}$$

$$z_j^k \in \{0, 1\}. \tag{19c}$$

The following states the formulation for a multilayer perceptron with K layers and n_k nodes j per layer. It

assumes the final output layer K to be singular and there not to be a ReLU function on that layer.

$$\begin{aligned}
\text{maximise} \quad & x_1^K && && (20a)\\
\text{subject to} \quad & \boldsymbol{W}^K \boldsymbol{x}^{K-1} + b^K = x_1^K && && (20b)\\
& \boldsymbol{W}_j^k \boldsymbol{x}^{k-1} + b_j^k = x_j^k - s_j^k && \forall k \in \{1,\dots,K-1\}, \forall j \in \{1,\dots,n_k\} && (20c)\\
& x_j^k \leq u_j^k z_j^k && \forall k \in \{1,\dots,K-1\}, \forall j \in \{1,\dots,n_k\} && (20d)\\
& s_j^k \leq -l_j^k (1 - z_j^k) && \forall k \in \{1,\dots,K-1\}, \forall j \in \{1,\dots,n_k\} && (20e)\\
& x_j^k, s_j^k \geq 0 && \forall k \in \{1,\dots,K-1\}, \forall j \in \{1,\dots,n_k\} && (20f)\\
& z_j^k \in \{0,1\} && \forall k \in \{1,\dots,K-1\}, \forall j \in \{1,\dots,n_k\} && (20g)\\
& x^0 \in \Omega && && (20h)
\end{aligned}$$

In this formulation, $\boldsymbol{W}_j^k$ is row j of the weight matrix of layer k, which naturally has the same dimension as the output $\boldsymbol{x}^{k-1}$ of the previous layer. The first input vector is constrained by the input constraints Ω. These are additional input constraints, containing the input bounds, but also other properties which can constrain the input vector, when used in surrogate models for example.

As noted by Grimstad and Andersson [17], this is an exact formulation of the ReLU neural network. This means that the above formulation exactly emulates the trained neural network from which the weight matrices W^k and biases b are extracted. For any given input x_0 the output of the MILP formulation and the neural net should have the same outcome. The solution which the MILP solver finds also finds consistent solution variables, with the exception of differing z_j^k variables in case the input node x_j^k is 0. This has no effect on the output however.

4 Methods

In this chapter the novel formulations of graph neural networks as MI(N)LPs are laid out, as an expansion on the multilayer perceptron MILP formulation as stated in Section 3.5. Section 4.1 describes an MILP formulation which is linear in case the graph structure is known. Section 4.2 describes a bi-linear MINLP formulation of the Graph Convolutional Network, in case the graph structure is unknown. Finally, Section 4.3 describes a linear MILP formulation of the GraphSAGE architecture.

4.1 Linear Formulation of Graph Neural Networks

In this thesis we focused on training a graph neural network which finds the function $f : \{0,1\}^{|N|\times|N|} \times \mathbb{R}^{|N|\times|F|} \mapsto \mathbb{R}$. This function maps an adjacency matrix A and a feature vector X, with $|F|$ features, to a singular output. The function is a composition of GNN layers, a pooling layer and MLP layers, where the latter takes a fingerprint of the graph and maps it to a singular output. Mathematically this constitutes to:

$$f(A, X) = \text{MLP}\ (\text{POOL}\ (\text{GNN}(A, X))) \tag{21}$$

where POOL is the pooling layer. This section states the linear MILP formulations for graph neural networks in case the graph structure is predetermined.

4.1.1 Predetermined Graph Structure

In the following section the Graph Convolutional Network layers as described in Section 3.4 are formulated as an MILP. To reiterate; a GCN layer as described by Kipf and Welling [7] has the following layer-wise propagation rule:

$$H^{(k+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(k)}W^{(k)}\right) \tag{22}$$

Here, $\tilde{A} = A + I_N$ is the adjacency matrix of the undirected graph $\mathcal{G}$ with added self-connections. I_N is the identity matrix, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ and $W^{(k)}$ is a layer-specific trainable weight matrix. $\sigma(\cdot)$ denotes an activation function, such as the $\text{ReLU}(\cdot) = \max(0, \cdot)$. $H^{(k)} \in \mathbb{R}^{N\times D}$ is the matrix of activations in the k^{th} layer; $H^{(0)} = X$, where X is a matrix of node feature vectors X_i belonging to node i in the graph.

To linearise Eq. (22) we use similar techniques as stated in the MILP formulation stated in Eqs (20). As described in Section 3.5, the ReLU constraints are the same for every node, with altering big-M values (lower and upper bounds). For the MLP structure, there were K layers with n_k neurons in each k^{th} layer. For the GCN structure we have the same but for every node $i \in \{1, \ldots, N\}$. In the first layer these input nodes are the feature vectors for those nodes.

Since the ReLU constraints stay the same, merely the left hand side of Eq. (20c) and Eq. (20d) need to be altered to represent the linear part of the GCN layer as described between the brackets in Eq. (22). To simplify the formulation we write $\bar{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$. The entries of this matrix are the following:

$$\bar{A}_{il} = \begin{cases} 0 & \text{if } \tilde{A}_{il} = 0 \\ \frac{1}{\sqrt{d^+(i)}\sqrt{d^+(l)}} & \text{if } \tilde{A}_{il} = 1 \end{cases} \tag{23}$$

where $d^+(i)$ is the cardinality of the adjacent set $N^+(i)$ for node i, where the $+$ indicates that it also includes self loops.

Kipf and Welling [7] note the following, with N nodes in our graph

$$Y^0 = X = \begin{bmatrix} X_1 \\ \dots \\ X_N \end{bmatrix} \tag{24}$$

where X_i is a row vector containing the features of node i. $H_{ij}^{(k)}$ is considered, which is the j-th neuron of node i, after k GCN layers. The value of this neuron is found as follows (the activation function σ is omitted from every line):

$$\begin{aligned}
H_{ij}^{(k)} &= \left(\bar{A}H^{(k-1)}W^{(k)}\right)_{ij} && \text{(25a)}\\
&= \left(\begin{bmatrix} \bar{A}_1 \\ \dots \\ \bar{A}_N \end{bmatrix}\right)_i \left(\begin{bmatrix} \left(H^{(k-1)}W^{(k)}\right)_1 & \dots & \left(H^{(k-1)}W^{(k)}\right)_N \end{bmatrix}\right)_j && \text{(25b)}\\
&= \bar{A}_i \left(\begin{bmatrix} H_1^{(k-1)} \\ \dots \\ H_N^{(k-1)} \end{bmatrix} \begin{bmatrix} W_1^{(k)} & \dots & W_N^{(k)} \end{bmatrix}\right)_j && \text{(25c)}\\
&= \bar{A}_i \begin{bmatrix} H_1^{(k-1)}W_j^{(k)} \\ \dots \\ H_N^{(k-1)}W_j^{(k)} \end{bmatrix} && \text{(25d)}\\
&= \sum_{l \in N^+(i)} \frac{1}{\sqrt{d^+(i)}\sqrt{d^+(l)}} H_l^{(k-1)}W_j^{(k)} && \text{(25e)}\\
&= \sum_{l \in N^+(i)} \frac{1}{\sqrt{d^+(i)}\sqrt{d^+(l)}} \left(W_j^{(k)}\right)^T \left(H_l^{(k-1)}\right)^T && \text{(25f)}
\end{aligned}$$

It is commonplace to write vectors in column notation for linear programming so we will deviate from Kipf and Welling [7], replacing $\left(H_l^{(k-1)}\right)^T$ by $\left(H_l^{(k-1)}\right)$. The MILP formulation becomes:

$$\sum_{l|\tilde{A}_{il}=1} \frac{1}{\sqrt{d_i^+ d_l^+}} \boldsymbol{W}_j^{k\,T} \boldsymbol{H}_l^{(k-1)} = H_{ij}^k - S_{ij}^k \qquad \forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{26a}$$

$$H_{ij}^k \leq U_{ij}^k Z_{ij}^k \qquad \forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{26b}$$

$$S_{ij}^k \leq -L_{ij}^k (1 - Z_{ij}^k) \qquad \forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{26c}$$

$$0 \leq H_{ij}^k, S_{ij}^k \in \mathbb{R} \qquad \forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{26d}$$

$$Z_{ij}^k \in \{0,1\} \qquad \forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{26e}$$

$$A, H^0 \in \Omega \tag{26f}$$

where i indicates node i, j feature j, and k the corresponding GCN layer. d_i^+ represents the degree $+1$ of node i. $H^0 \in \Omega$ indicates a restriction of the input space (see Section 4.4). In case of the molecule case we can see these as constraints such that only physically possible molecules are considered in the input space. One example could be that node i, has a feature $x_{ij} = 1$ indicating that it is a carbon molecule. In that case $\sum_j A_{ij} < 5$, meaning it can't share a bond with more than 4 other molecules since the amount of atomic bonds is maximally 4 for a carbon molecule.

Combining all constraints of (26) with (20) almost finalises our formulation of function (21) in case the graph structure A is known. We still need to include a pooling layer. The pooling layer is a function which operates over all neuron outputs of the nodes of the graph to combine them into a single vector. There are multiple options for pooling layers but we utilise a sum pooling layer. The sum pooling layer takes all neuron outputs of each node and sums them. Since all neuron outputs have an equal length this operation can be performed. This creates what's called a fingerprint of the graph, which serves as an input for the MLP layers. The following constraint emulates the chosen pooling function:

$$x_j^0 = \sum_{i=1}^{N} H_{ij}^K \quad \forall j \in \{1, \dots, n_K\} \tag{27}$$

Naturally, the amount of entries of the input vector x_0 of the MLP layer, must match the number of neurons in the final layer K of the GCN layers.

4.1.2 Linearising the Non-Linear Terms

The formulation of the previous section is linear in case the structure of the graph is known. If A is unknown, this formulation is non-linear. There are many examples where we wish to find the graph structure accompanying an optimal solution. The non-linear terms in the above formulation, in case the structure is unknown, are $\sum_{l|\tilde{A}_{il}=1}$ and $\frac{1}{\sqrt{d_i^+ d_l^+}}$, where l in the latter is dependant on the former non-linear term. Steps can be taken to make these non-linear terms linear.

A linear conditional sum

The sum in the formulation is conditional and thus non-linear. To linearise the sum a support variable $\boldsymbol{b}_{il}^k$ is introduced. This new support variable follows the following logic for two nodes i and j:

$$\boldsymbol{b}_{il}^k = \begin{cases} 0 & \text{if } \tilde{A}_{il} = 0 \\ \boldsymbol{H}_l^k & \text{if } \tilde{A}_{il} = 1 \end{cases} \tag{28}$$

If this logic is implemented the sum over all $\boldsymbol{b}_{il}^k$ results in the same outcome as the conditional sum. For every node i, $\boldsymbol{b}_{il}^k$ are only equal to the output of ReLU layer if they are connected to node i.

The logic as described in (28) can be implemented in the same way as was done in Eq. (18) by using big-M constraints, because the entries of $\tilde{A}$ are binary. Replacing Eq. (26a) by the following constraints for $k \in \{1, \dots, K\}, i \in \{1, \dots, N\}, j \in \{1, \dots, n_k\}$ we have removed the conditional sum:

$$\sum_l \frac{1}{\sqrt{d_i^+ d_l^+}} \boldsymbol{W}_j^{k\,T} \boldsymbol{b}_{il}^{(k-1)} = H_{ij}^k - S_{ij}^k \tag{29a}$$

$$\boldsymbol{H}_l^{(k-1)} - \boldsymbol{M}(1 - \tilde{A}_{il}) \leq \boldsymbol{b}_{il}^{(k-1)} \leq \boldsymbol{H}_l^{(k-1)} + \boldsymbol{M}(1 - \tilde{A}_{il}) \tag{29b}$$

$$-\boldsymbol{M}(\tilde{A}_{il}) \leq \boldsymbol{b}_{il}^{(k-1)} \leq \boldsymbol{M}(\tilde{A}_{il}) \tag{29c}$$

In case node i is not connected to node l, then $\tilde{A}_{il} = 0$. In that case constraint (29c) forces $\boldsymbol{b}_{il}^{(k-1)} = 0$. In case both nodes are connected, $\tilde{A}_{il} = 1$ and $\boldsymbol{b}_{il}^{(k-1)}$ is constrained by (29b) such that it is equal to $\boldsymbol{H}_l^{(k-1)}$.

A linear normalisation term

We are still left with $\frac{1}{\sqrt{d_i^+ d_l^+}}$, which is also non-linear. The term is also multiplied with the variable vector b_{il}^k,

which makes the entire constraint non-linear. It is possible to remove the fraction and the square root, albeit in a contrived way, adding a lot of extra variables. We note that the cardinality of the co-domain of the function $g(i,l) = \frac{1}{\sqrt{d_i^+ d_l^+}}$, is upper bound by the maximum degree of the graph d_{max}, adding 1 for the self loops. In case of molecules this is 4 for instance. This means that the function g has a maximum of $(4+1)^2$ outcomes. We can index these outcomes in a $(d_{max}+1)^2$ long vector g, where at index $p = d_i^+(d_{max}+1) + d_l^+$, $g_p = \frac{1}{\sqrt{d_i^+ d_l^+}}$. The function $g(i,l)$ is undefined in case $d_i^+ = 0$ or $d_l^+ = 0$. In these cases $g_p = 0$. Using linear constraints, we can linearize the fractional term in Eq. (29a) by the following set of equations

$$\sum_l s_{il} {\boldsymbol{W}_j^k}^T \boldsymbol{b}_{il}^{(k-1)} = H_{ij}^k - S_{ij}^k \qquad \forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \quad (30a)$$

$$d_i^+ = \sum_j \tilde{A}_{ij} \qquad \forall i \in \{1,\dots,N\} \quad (30b)$$

$$p_{il} = d_i^+(d_{max}+1) + d_l^+ \qquad \forall i,l \in \{1,\dots,N\} \quad (30c)$$

$$0 = p_{il} - 1c_1^{il} - 2c_2^{il} - \cdots - (d_{max}+1)^2 c_{(d_{max}+1)^2}^{il} \qquad \forall i,l \in \{1,\dots,N\} \quad (30d)$$

$$1 = c_1^{il} + \cdots + c_{(d_{max}+1)^2}^{il} \qquad \forall i,l \in \{1,\dots,N\} \quad (30e)$$

$$c^{il} \in \{0,1\}^{(d_{max}+1)^2} \qquad \forall i,l \in \{1,\dots,N\} \quad (30f)$$

$$s_{il} = c_1^{il} g_1 + \cdots + c_{(d_{max}+1)^2}^{il} g_{(d_{max}+1)^2} \qquad \forall i,l \in \{1,\dots,N\} \quad (30g)$$

With this set of equations, we are mapping the index p_{il} to its corresponding value in vector g, which is a set of predetermined parameters. Since the structure of graph stays the same over all GCN layers, we only have to add these constraints once and not for every layer k.

As can be seen in Eqs. (30), there still exists a non-linear multiplication of decision variables s_{il} and b_{il}. This multiplication is a special non-linear case, namely a bi-linear term. This problem belongs to a class of MINLPs for which effective methods in commercial solver exist to find optimality. In the next section we take the linearisations described in this section and apply them to make an MINLP formulation of the GCN model.

4.2 MINLP Formulation of the GCN GNN

In this section the bi-linear formulation is described. To introduce the bi-linear formulation we start again at the MILP formulation of the MLP layers, pooling layer and GCN layers. These combination of constraints is laid out in Eqs. (20), (26) and (27). As described before, it is the goal to simplify Eq. (26a) which we will restate for the clarity of the argumentation:

$$\sum_{l|\tilde{A}_{il}=1} \frac{1}{\sqrt{d_i^+ d_l^+}} \boldsymbol{W}_j^{k^T} \boldsymbol{H}_l^{(k-1)} = H_{ij}^k - S_{ij}^k \qquad \forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{31}$$

In Section 4.1.2 it is laid out how to linearise non-linear terms of Eq. (31), which are the conditional sum and the normalisation term. This resulted in a bi-linear formulation as described by the constraints in Eqs. (29b), (29c) and (30). Notice how for every layer k, Eqs. (29b) and (29c) constrain whether or not the feature vector of the neighbours of node i are included in the conditional sum. We can simplify this by incorporating it in the variable which encompasses the linearised normalisation term s_{il}. Once again we incorporate the following logic with big-M constraints for $i, l \in \{1,\dots,N\}$:

$$\hat{s}_{il} = \begin{cases} 0 & \text{if } \tilde{A}_{il} = 0 \\ s_{il} & \text{if } \tilde{A}_{il} = 1 \end{cases} \tag{32}$$

This makes it such that the feature vector of a neighbouring node of i only gets added in case $\tilde{A}_{il} = 1$, which is the same as $\sum_{l|\tilde{A}_{il}=1}$. The resulting bi-linear MINLP formulation becomes:

$$\sum_l \hat{s}_{il} \boldsymbol{W}_j^{k^T} \boldsymbol{H}_l^{(k-1)} = H_{ij}^k - S_{ij}^k \qquad \forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{33a}$$

$$d_i^+ = \sum_j \tilde{A}_{ij} \qquad \forall i \in \{1,\dots,N\} \tag{33b}$$

$$p_{il} = d_i^+ (d_{max} + 1) + d_l^+ \qquad \forall i, l \in \{1,\dots,N\} \tag{33c}$$

$$0 = p_{il} - 1c_1^{il} - 2c_2^{il} - \cdots - (d_{max}+1)^2 c_{(d_{max}+1)^2}^{il} \qquad \forall i, l \in \{1,\dots,N\} \tag{33d}$$

$$1 = c_1^{il} + \cdots + c_{(d_{max}+1)^2}^{il} \qquad \forall i, l \in \{1,\dots,N\} \tag{33e}$$

$$c^{il} \in \{0,1\}^{(d_{max}+1)^2} \qquad \forall i, l \in \{1,\dots,N\} \tag{33f}$$

$$s_{il} = c_1^{il} g_1 + \cdots + c_{(d_{max}+1)^2}^{il} g_{(d_{max}+1)^2} \qquad \forall i, l \in \{1,\dots,N\} \tag{33g}$$

$$s_{il} - M(1 - A_{il}) \leq \hat{s}_{il} \leq M(1 - A_{il}) + s_{il} \qquad \forall i, l \in \{1,\dots,N\} \tag{33h}$$

$$-MA_{il} \leq \hat{s}_{il} \leq MA_{il} \qquad \forall i, l \in \{1,\dots,N\} \tag{33i}$$

In this formulation, constraints (33b) - (33g) describe the linearisation of the normalisation term as described in Section 4.1.2 and constraints (33h) and (33i) incorporate the logic from Eq. (32). Notice that we only have to find $\hat{s}_{il}$ once for every layer, since the structure of the molecule doesn't change per layer.

A reader might note that when the degree of either node i or node j is zero, this means that s_{il} will automatically be equal to zero, and thus the introduction of Eqs. (33h) and (33i) might be superfluous. However, there could be an instance when both i and l have a degree higher than 0, but still not be connected. In that case s_{il} is not zero, and thus the extra constraints need to be introduced.

4.3 GraphSAGE

In this section we describe a different GNN architecture which allows us to propose a completely linear MILP formulation of a GNN. The GNN model we chose to linearise is the *GraphSAGE* model by Hamilton et al. [8]. In this paper the activation function and affine layer are described by the following equation:

$$f^{(t)}(v) = \sigma\left(f^{(t-1)}(v) \cdot W_1^{(t)} + \sum_{w \in N(v)} f^{(t-1)}(w) \cdot W_2^{(t)}\right) \tag{34}$$

where $f^{(t)}(v)$ describes the feature vector of node v after t GraphSAGE layers. As before σ describes an activation function, which for the purpose of this thesis will once again be the ReLU activation function.

As can be seen from Eq. (34), after training a neural net with K GraphSAGE layers, it finds two weight matrices for every layer t. The first weight matrix $W_1^{(t)}$, which we will refer to as the root weight, is multiplied with the feature vector of the previous layer $f^{(t-1)}(v)$. The second weight matrix $W_2^{(t)}$ is multiplied with the neighbouring feature vectors of node v. In case the adjacency matrix of the graph is unknown, this neighbourhood of v is a non-linear relation. The rest is linear however. In the next paragraph we describe how to linearise this conditional sum.

For consistency's sake we rewrite Eq. (34) in a notation similar to the rest of this thesis. We find the following for node $i \in \{1, \dots, N\}$, feature $j \in \{1, \dots, n_k\}$ and layer $k \in \{1, \dots, K\}$:

$$H_{ij}^{(k)} = \sigma\left(\hat{\boldsymbol{W}}_j^{kT} \boldsymbol{H}_i^{(k-1)} + \bar{\boldsymbol{W}}_j^{kT} \sum_{l | A_{il}=1} \boldsymbol{H}_l^{(k-1)}\right) \tag{35}$$

where $H_{ij}^{(k)} \in \mathbb{R}$ is the feature j of node i after k layers, and A_{il} is the adjacency matrix without self loops. To remove the conditional sum we again introduce big-M constraints and support variables to encode the following logic:

$$\boldsymbol{b}_{il}^k = \begin{cases} 0 & \text{if } A_{il} = 0 \\ \boldsymbol{H}_l^k & \text{if } A_{il} = 1 \end{cases} \tag{36}$$

The full MILP formulation including the pooling layer becomes:

$$(\hat{\boldsymbol{W}}_j^k)^T \boldsymbol{H}_i^{(k-1)} + (\bar{\boldsymbol{W}}_j^k)^T \sum_l \boldsymbol{b}_{il}^{(k-1)} = H_{ij}^k - S_{ij}^k \quad \forall k \in \{1, \dots, K\}, \forall i \in \{1, \dots, N\}, \forall j \in \{1, \dots, n_k\} \tag{37a}$$

$$H_{ij}^k \leq U_{ij}^k Z_{ij}^k \quad \forall k \in \{1, \dots, K\}, \forall i \in \{1, \dots, N\}, \forall j \in \{1, \dots, n_k\} \tag{37b}$$

$$S_{ij}^k \leq -L_{ij}^k (1 - Z_{ij}^k) \quad \forall k \in \{1, \dots, K\}, \forall i \in \{1, \dots, N\}, \forall j \in \{1, \dots, n_k\} \tag{37c}$$

$$\boldsymbol{H}_l^k - \boldsymbol{M}(1 - A_{il}) \leq \boldsymbol{b}_{il}^k \quad \forall k \in \{0, \dots, K-1\}, \forall i, l \in \{1, \dots, N\} \tag{37d}$$

$$\boldsymbol{b}_{il}^k \leq \boldsymbol{H}_l^k + \boldsymbol{M}(1 - A_{il}) \quad \forall k \in \{0, \dots, K-1\}, \forall i, l \in \{1, \dots, N\} \tag{37e}$$

$$-\boldsymbol{M}(A_{il}) \leq \boldsymbol{b}_{il}^k \leq \boldsymbol{M}(A_{il}) \quad \forall k \in \{0, \dots, K-1\}, \forall i, l \in \{1, \dots, N\} \tag{37f}$$

$$\boldsymbol{H}_i^0 = \boldsymbol{x}_i \quad \forall i \in \{1, \dots, N\} \tag{37g}$$

$$H_j^{*K} = \sum_i H_{ij}^K \quad \forall j \in \{1, \dots, n_K\} \tag{37h}$$

$$0 \leq H_{ij}^k, S_{ij}^k \in \mathbb{R} \quad \forall k \in \{1, \dots, K\}, \forall i \in \{1, \dots, N\}, \forall j \in \{1, \dots, n_k\} \tag{37i}$$

$$Z_{ij}^k \in \{0, 1\} \quad \forall k \in \{1, \dots, K\}, \forall i \in \{1, \dots, N\}, \forall j \in \{1, \dots, n_k\} \tag{37j}$$

$$\boldsymbol{x}, A \in \Omega \tag{37k}$$

To clarify, Eqs. (37a - 37c) linearise the ReLU function in Eq. (35), Eqs. (37d - 37f) are the big-M constraints to force the logic in Eq. (36). The values of these big-M constraints are the same as the upper bounds U_{ij}^k (as explained in Section 4.5.3). Eq. (37g) defines the input feature vector $\boldsymbol{x}_i$ of node i and Eq. (37h) is the sum pooling layer in layer K. Finally, Eq. (37k) represent the input constraints as described in Section 4.4.

Note that a drawback of this method compared to the MINLP formulation is that constraints (37d), (37e) and (37f) are calculated for every layer k. This increases the amount of constraints significantly. Namely, $\mathcal{O}(n^2 n_k)$ constraints per layer k. For the GCN network this is only $\mathcal{O}(n^2)$. For networks where there are a lot of nodes per hidden layer, the GraphSAGE network will have more constraints than the GCN network.

4.4 Constraining the Input Space for Molecular Design

In this section we describe the input space constraints referred in the thesis as $A, x \in \Omega$. These constraints limit the search space to include structures which try to emulate molecular structures.

4.4.1 Basic MILP formulation of Molecules

QSPR methods used for property prediction in previous works are mostly based on group contribution methods [97]. As a result MILP formulations of molecules used in CAMD are also often based on group contribution methods [97]. Modelling chemical properties with GNNs means that molecules are described in terms of an adjacency matrix $A \in \{0,1\}^{N\times N}$ and feature vectors $X \in \{0,1\}^{N\times F}$. We therefor introduce an MILP formulation for molecules based on the structure similar to the input of GNNs.

The structure of a solution is described by the adjacency matrix A, where $A_{ij} = 1$ indicates that node i is connected to node j. The entries of a feature vector of a node i are indicated by x_{if}, where f is the position of a feature in that vector. The simplest machine learning model that was considered consists of 14 features. These features represent the following:

X_{if}	type	descriptor	X_{if}	group	descriptor	X_{if}	group	descriptor
1	atom	C	6	neighbours	1	11	hydrogen	1
2	atom	O	7	neighbours	2	12	hydrogen	2
3	atom	F	8	neighbours	3	13	hydrogen	3
4	atom	Cl	9	neighbours	4	14	hydrogen	4
5	neighbours	0	10	hydrogen	0			

With the adjacency matrix and the feature vectors for all the nodes, we introduce the following set of constraints:

$$A_{11} = A_{22} = A_{12} = 1 \tag{38a}$$

$$A_{ii} \geq A_{(i+1),(i+i)} \qquad \forall i \in \{1,\dots,n-1\} \tag{38b}$$

$$A_{ii} = x_{i,1} + \cdots + x_{i,4} \qquad \forall i \in \{1,\dots,n\} \tag{38c}$$

$$A_{ii} = x_{i,5} + \cdots + x_{i,9} \qquad \forall i \in \{1,\dots,n\} \tag{38d}$$

$$A_{ii} = x_{i,10} + \cdots + x_{i,14} \qquad \forall i \in \{1,\dots,n\} \tag{38e}$$

$$\begin{aligned} &4x_{i,1} + 2x_{i,2} + 1x_{i,3} + 1x_{i,4} = \\ &\quad 0x_{i,5} + 1x_{i,6} + 2x_{i,7} + 3x_{i,8} + 4x_{i,9} \\ &\quad + 0x_{i,10} + 1x_{i,11} + 2x_{i,12} + 3x_{i,13} + 4x_{i,14} \end{aligned} \qquad \forall i \in \{1,\dots,n\} \tag{38f}$$

$$\sum_{j,i\neq j} A_{ij} = 0x_{i,5} + 1x_{i,6} + 2x_{i,7} + 3x_{i,8} + 4x_{i,9} \qquad \forall i \in \{1,\dots,n\} \tag{38g}$$

$$A_{ij} = A_{ji} \qquad \forall i,j \in \{1,\dots,n\} \tag{38h}$$

$$M_4 A_{ii} \geq \sum_{j,i\neq j} A_{ij} \qquad \forall i \in \{1,\dots,n\} \tag{38i}$$

$$A_{ii} \leq \sum_{j,i\neq j} A_{ij} \qquad \forall i \in \{1,\dots,n\} \tag{38j}$$

$$M_5 A_{ii} \geq \sum_{f=1}^{14} x_{if} \qquad \forall i \in \{1, \ldots, n\} \tag{38k}$$

$$A_{ii} \leq \sum_{j<i} A_{ij} \qquad \forall i \in \{3, \ldots, n\} \tag{38l}$$

where n is the number of atoms in the molecule, the big-M values are defined as $M_4 = n+1$ and $M_5 = |F| = 14$. One general remark, we can think of the atom i to be 'on' when $A_{ii} = 1$. Next is a summation including an explanation for all constraints.

(a) We always want molecules of at least length 2 and these are also connected (see point (l))

(b) This is a symmetry braking constraint. We want no gaps between activated atoms. So if we want an molecule that is three atoms long, we want $A_{11} = A_{22} = A_{33} = 1$ and not $A_{11} = A_{22} = A_{55} = 1$. This constraint prevents that.

(c) We only want one atom type

(d) We only want one number of neighbours to be indicated

(e) We only want one number of hydrogen neighbours to be indicated

(f) The covalence of the atom must equal the number of neighbours + the number of hydrogen neighbours. Let's say we have a carbon atom on position 1, then the covalence is 4. If it is connected to two other atoms, $x_{1,7} = 1$ and thus $x_{1,12}$ must be equal to one too.

(g) The number of neighbours in the feature vector must equal the out degree in the adjacency matrix

(h) The adjacency matrix is non directed, and thus symmetric. This constraint helps gurobi decrease the amount of possibilities it has to search for.

(i) This is a big-M constraint. It forces for atom i that $A_{ii} = 1$ if it is connected to any other atoms. We only want an atom to be active if it is connected to others.

(j) This constraint forces the opposite of the previous constraint, namely that if an atom is not connected to any other atoms it should be deactivated.

(k) This is the last big-M constraint. It forces the feature vectors of atom i to be 0 in case $A_{ii} = 0$.

(l) This constraints forces no disconnected sub graphs. An example is that you indicate to the program that you want to find a molecule which is 4 atoms long. Without this constraint it could find two molecules, namely $H_3C - CH_3$ and $H_3C - OH$. However, we want to find a single molecule.

To combat this we make it such that every atom that is activated is connected to one of the previously activated atoms. For instance, if $A_{44} = 1$, then either A_{41}, A_{42} or A_{43} needs to equal 1. This forces a connected graph.

This constraint does remove molecule configurations from the search space. Consider the following example. Let's say we have the following adjacency matrix for a molecule $CH_3 - O - CH_2 - CH_3$:

$$\text{A} = \begin{array}{|c|c|c|c|} \hline 1 & 1 & 0 & 1 \\ \hline 1 & 1 & 0 & 0 \\ \hline 0 & 0 & 1 & 1 \\ \hline 1 & 0 & 1 & 1 \\ \hline \end{array}$$

This case could not be found due to constraint (38l), however it is a feasible molecule. The molecule can be found another configuration that is congruent with (38l), namely:

$$A^* = \begin{array}{|c|c|c|c|}\hline 1 & 1 & 0 & 0 \\ \hline 1 & 1 & 1 & 0 \\ \hline 0 & 1 & 1 & 1 \\ \hline 0 & 0 & 1 & 1 \\ \hline \end{array}$$

However, the objective values of this molecule with the different adjacency matrices is not the same, due to the non symmetric weight matrices.

The above formulation is not a tight formulation and molecules can be found in the search space that might not be able to be synthesised or stable in a natural setting. For instance, the below formulation does not consider steric constraints on the bonds. There are also molecules which are excluded from the search space. An example of these are molecules with double or triple bonds. In the next section we describe how to extend this model.

4.4.2 Extra Properties

The formulation in the previous section should be seen as a basis which can be extended. Additional constraints can be introduced such that the search space of possible molecules is catered to the needs of a researcher who would like to use this model.

First of all, to limit the search space to molecules with no loops, consider the following constraint:

$$\sum_{i}^{n-1} \sum_{j|j>i}^{n} A_{ij} = n - 1 \tag{39}$$

The constraint guarantees that the amount of edges (LHS) is equal to the amount of nodes - 1. When added to the set of constraints (38), no loops will be present in the molecules in the search space. Before adding the extra constraint, the search space only includes connected graphs due to constraint (38l). This fact, combined with corollary 3.1.1 and the result of constraint (39), guarantees the graph to be acyclic.

Next we consider constraints to have the search space include double bonded molecules. To achieve double bonds in our formulation, an extra feature is included in the feature vectors $X \in \mathbb{R}^{N \times F}$. This feature $x_{i,15}$ indicates whether node i is included in at least one double bond. This is a learnable parameter for the GNN. Outside the context of MILP formulations for GNNs, this feature would be included in a GNN which also includes edge features. However, this thesis doesn't consider these network architectures.

To complete the inclusion of double bonded molecules in the search space the following set of constraints should be included. Also, a new binary variable db_{il} is introduced which indicates that a double bond is present between node i and l.

$$3 * db_{il} \leq x_{i,15} + x_{l,15} + A_{il} \qquad \forall i, l \in \{1, \dots, n\} \tag{40a}$$

$$2 * x_{i,1} + 1 * x_{i,2} \geq \sum_{l}^{n} db_{il} \qquad \forall i \in \{1, \dots, n\} \tag{40b}$$

$$4 * x_{i,1} + 2 * x_{i,2} + 1 * x_{i,3} + 1 * x_{i,4} = \sum_{s=0}^{4} s * x_{i,(5+s)} + \sum_{s=0}^{4} s * x_{i,(10+s)} + \sum_{l}^{n} db_{il} \qquad \forall i \in \{1, \dots, n\} \tag{40c}$$

$$x_{i,15} \leq \sum_{l}^{n} db_{il} \qquad \forall i \in \{1, \dots, n\} \tag{40d}$$

$$db_{il} = db_{li} \qquad \forall i, l \in \{1, \dots, n\} \tag{40e}$$

$$db_{ii} = 0 \qquad \forall i \in \{1, \dots, n\} \tag{40f}$$

Constraint (40a) makes it such that a double bond between node i is only possible if there features $x_{i,15}$ and $x_{l,15}$ are on, and there is a connection between node i and l. Constraint (40b) limits the amount of double bonds based on the covalence of the molecule. For instance, $x_{i,1}$ indicates that there is a carbon molecule, meaning that there can be a maximum of 2 double bonds. Constraint (40c) is the same as constraint (38f) plus counting how many double bonds node i is connected to. For every double bond, the amount of hydrogen neighbours gets reduced. Constraint (40d) forces $x_{i,15}$ to be zero in case there are no double bonds connected to node i. The reasoning is that activating feature $x_{i,15}$ might have positive effects on the overall objective value of the MILP. In case no double bonds are in the solution, this means that $x_{i,15}$ should also not be on. The final two constraints are a symmetry constraint and a constraint indicating that a node cannot have a double bond with itself.

For triple bonds, the above formulation would be nearly identical. Instead, the variable db_{il} would be replaced by tb_{il} in all constraints to indicate a triple bond between node i and node l. Constraint (40b) would have $1 * x_{i,1}$ on the left hand side because generally, only a carbon molecule can have a triple bond. Finally, in constraint (40c), $\sum_l^n tb_{il}$ would include a scalar multiple of 2, because every triple bond removes two binding opportunities.

Including both triple and double bonds in the search space can be achieved with the following set of constraints.

$$3 * db_{il} \leq x_{i,15} + x_{l,15} + A_{il} \qquad \forall i, l \in \{1, \dots, n\} \tag{41a}$$

$$3 * tb_{il} \leq x_{i,16} + x_{l,16} + A_{il} \qquad \forall i, l \in \{1, \dots, n\} \tag{41b}$$

$$2 * x_{i,1} + 1 * x_{i,2} \geq \sum_{l}^{n} db_{il} \qquad \forall i \in \{1, \dots, n\} \tag{41c}$$

$$1 * x_{i,1} \geq \sum_{l}^{n} tb_{il} \qquad \forall i \in \{1, \dots, n\} \tag{41d}$$

$$\begin{aligned} 4 * x_{i,1} + 2 * x_{i,2} + 1 * x_{i,3} + 1 * x_{i,4} = \sum_{s=0}^{4} s * x_{i,(5+s)} + \sum_{s=0}^{4} s * x_{i,(10+s)} \\ + \sum_{l}^{n} db_{il} + \sum_{l}^{n} 2 * tb_{il} \end{aligned} \qquad \forall i \in \{1, \dots, n\} \tag{41e}$$

$$x_{i,15} \leq \sum_{l}^{n} db_{il} \qquad \forall i \in \{1, \dots, n\} \tag{41f}$$

$$x_{i,16} \leq \sum_{l}^{n} tb_{il} \qquad \forall i \in \{1, \dots, n\} \tag{41g}$$

$$db_{il} = db_{li} \qquad \forall i, l \in \{1, \dots, n\} \tag{41h}$$

$$tb_{il} = tb_{li} \qquad \forall i, l \in \{1, \dots, n\} \tag{41i}$$

$$db_{ii} = tb_{ii} = 0 \qquad \forall i \in \{1, \dots, n\} \tag{41j}$$

$$db_{li} + tb_{li} \leq 1 \qquad \forall i, l \in \{1, \dots, n\} \tag{41k}$$

Where $x_{i,15}$ and $x_{i,16}$ indicate that an atom i is part of a double or triple bond respectively.

Once again we note that the introduction of these constraints doesn't span the entire space of possible molecules, nor does it include only naturally feasible molecules. For instance, introducing the triple bonds constraints would not find the molecule carbon monoxide.

4.5 Bound Tightening Techniques

Solving times of linear programming solvers are influenced by the tightness of the big-M constraints. It is therefore important to find tight constraints of the big-M values associated with a neuron. We first take a look at the computationally efficient method of feasibility based bound tightening (FBBT). We first consider this for regular MLPs and then we continue adapting these methods for the GCN and GraphSAGE.

4.5.1 MLPs

Feasibility based bound tightening techniques are bound tightening techniques which limit the feasible solution space by propagating the domain of the input space through the non-linear expression. This technique relies on interval arithmetic to compute the bounds on constraint activations over the variable domains [98]. For the MILP formulation of MLPs we note the following constraints to implement the ReLU function:

$$\boldsymbol{W}_j^k \boldsymbol{x}^{k-1} + b_j^k = x_j^k - s_j^k \qquad \forall k \in \{1, \dots, K-1\}, \forall j \in \{1, \dots, n_k\} \tag{42a}$$

$$x_j^k \leq u_j^k z_j^k \qquad \forall k \in \{1, \dots, K-1\}, \forall j \in \{1, \dots, n_k\} \tag{42b}$$

$$s_j^k \leq -l_j^k (1 - z_j^k) \qquad \forall k \in \{1, \dots, K-1\}, \forall j \in \{1, \dots, n_k\} \tag{42c}$$

Using interval arithmetic we can find bounds for the nodes for $k \geq 2$ in two ways, which result in the same bounds. For the first method, for layers $k \geq 2$ we find the upper bound u_j^k and lower bound l_j^k as follows for a node j in layer k:

$$u_j^k = \sum_{i=1}^{n_{k-1}} \max\left\{w_{ji}^k \max\left\{0, u_i^{k-1}\right\}, w_{ji}^k \max\left\{0, l_i^{k-1}\right\}\right\} + b_j^k, \tag{43a}$$

$$l_j^k = \sum_{i=1}^{n_{k-1}} \min\left\{w_{ji}^k \max\left\{0, u_i^{k-1}\right\}, w_{ji}^k \max\left\{0, l_i^{k-1}\right\}\right\} + b_j^k. \tag{43b}$$

Note that the inner max functions function as the ReLU activation function. The outer max function is necessary since the weight matrix entries can also be negative. For $k = 1$ we remove the inner max functions as the input is not necessarily positive since there is no ReLU operator. The same bounds can be found by solving the LP problems:

$$\begin{aligned} u_j^k &= \max\left\{t_j^k : t_j^k \in C_j^k\right\} \\ l_j^k &= \min\left\{t_j^k : t_j^k \in C_j^k\right\} \end{aligned} \tag{44}$$

for the constraint set

$$C_j^k = \left\{t_j^k : t_j^k = w_j^k x^{k-1} + b^k, x^{k-1} \in \left[\max\left\{0, L^{k-1}\right\}, \max\left\{0, U^{k-1}\right\}\right]\right] \quad \subset \mathbb{R}^{n_{k-1}}\} \tag{45}$$

To speed up the solving time, some activation variables z can be determined based on the value of the lower and upper bound. When the lower bound l_j^k of a particular node is above 0, z_j^k can be set to 1. In this case it is known that x_j^k will always be positive and thus z_j^k needs to be 1 to satisfy constraint (42b). The same goes for a positive lower bound $l_j^k > 0$. In this case $z_j^k = 0$.

4.5.2 GCN

The following section explains how to find the upper and lower bound associated with the ReLU constraints for a GCN model. The constraints for which we want to find the lower and upper bound are the following:

$$\sum_{l|\tilde{A}_{il}=1} \frac{1}{\sqrt{d_i^+ d_l^+}} \boldsymbol{W}_j^{k\,T} \boldsymbol{H}_l^{(k-1)} = H_{ij}^k - S_{ij}^k \qquad \forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{46a}$$

$$H_{ij}^k \leq U_{ij}^k Z_{ij}^k \qquad \forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{46b}$$

$$S_{ij}^k \leq -L_{ij}^k (1 - Z_{ij}^k) \qquad \forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\}. \tag{46c}$$

It is assumed that the lower and upper bound of all the input feature vectors are the same. This is because the input feature vectors of all nodes describe the same features of those nodes, and for any graph neural network to work, the feature vectors need to equal in length. This makes the bound propagation symmetric over all nodes, and this allows us to only calculate the bounds of all nodes once per layer k. Before the optimisation, for node i, the number of neighbouring nodes and the number of their respective neighbours are unknown. In case of maximisation we have to find a scalar which upper bounds H_{ij}^k for all possible neighbourhood structures of node i. This means we have to d_i^+ and d_l^+ such the following is maximised (remember that $d_i^+ = d_i + 1$, where d_i is the degree of node i if self loops are not possible):

$$d_i^+ \frac{1}{\sqrt{d_i^+ d_l^+}} \boldsymbol{W}_j^{k\,T} \boldsymbol{H}_l^{(k-1)} \tag{47}$$

The upper bound of $\boldsymbol{W}_j^{k\,T} \boldsymbol{H}_l^{(k-1)}$ is determined as in Eq. (43a) with zero bias. In case the outcome of this upper bound is positive, we want to add as much as possible to account for all possible neighbourhood structures. This means $\frac{\sqrt{d_i^+}}{\sqrt{d_l^+}}$ needs to be maximised. Naturally we maximise the numerator en minimise the denominator. The degree of node i, d_i^+, can maximally be $d_{max} + 1$, and the minimum degree of the neighbours of i needs to be $d_l^+ = 1 + 1$. In case that the upper bound determined by Eq. (43a) is negative $\frac{\sqrt{d_i^+}}{\sqrt{d_l^+}}$ needs to be minimal, to find a maximum. Determining the lower bound follows the same logic. This results in

$$\begin{aligned} U_{ij}^k = \max\Bigg\{ & \frac{\sqrt{d_{max}+1}}{\sqrt{2}} \sum_{s=1}^{n_{k-1}} \max\left\{w_{js}^k \max\left\{0, U_{sj}^{k-1}\right\}, w_{js}^k \max\left\{0, L_{sj}^{k-1}\right\}\right\}, \\ & \frac{\sqrt{2}}{\sqrt{d_{max}+1}} \sum_{s=1}^{n_{k-1}} \max\left\{w_{js}^k \max\left\{0, U_{sj}^{k-1}\right\}, w_{js}^k \max\left\{0, L_{sj}^{k-1}\right\}\right\} \Bigg\} \end{aligned} \tag{48a}$$

$$\begin{aligned} L_{ij}^k = \min\Bigg\{ & \frac{\sqrt{d_{max}+1}}{\sqrt{2}} \sum_{s=1}^{n_{k-1}} \min\left\{w_{js}^k \max\left\{0, U_{sj}^{k-1}\right\}, w_{js}^k \max\left\{0, L_{sj}^{k-1}\right\}\right\}, \\ & \frac{\sqrt{2}}{\sqrt{d_{max}+1}} \sum_{s=1}^{n_{k-1}} \min\left\{w_{js}^k \max\left\{0, U_{sj}^{k-1}\right\}, w_{js}^k \max\left\{0, L_{sj}^{k-1}\right\}\right\} \Bigg\} \end{aligned} \tag{48b}$$

In practice we only need the terms multiplied with $\frac{\sqrt{d_{max}+1}}{\sqrt{2}}$ in both u_{ij}^k and l_{ij}^k. This is because in the case that $\max\left\{w_{ji}^k \max\left\{0, u_i^{k-1}\right\}, w_{ji}^k \max\left\{0, l_i^{k-1}\right\}\right\}$ is negative, the node is turned off and on for the upper bound and lower bound respectively (as described in Section 4.5.1).

For a similar formulation as Eqs. (44) and (45), the following set of equations can be considered:

$$U_{ij}^k = \max\left\{t_{ij}^k : t_{ij}^k \in C_{ij}^k\right\} \tag{49a}$$

$$L_{ij}^k = \min\left\{t_{ij}^k : t_{ij}^k \in C_{ij}^k\right\} \tag{49b}$$

$$C_{ij}^k = \left\{t_{ij}^k : t_{ij}^k = \frac{\sqrt{d_{max}+1}}{\sqrt{2}} w_j^k x^{k-1}, x^{k-1} \in \left[\max\left\{0, L_i^{k-1}\right\}, \max\left\{0, U_i^{k-1}\right\}\right] \subset \mathbb{R}^{n_{k-1}}\right\} \tag{49c}$$

4.5.3 GraphSAGE

For GraphSAGE, FBBT is very similar to the FBBT proposed for the GCN. We are trying to find bounds for $k \in \{1,\dots,K\}, i \in \{1,\dots,N\}, j \in \{1,\dots,n_k\}$ for the following set of equations:

$$(\hat{\boldsymbol{W}}_j^k)^T \boldsymbol{H}_i^{(k-1)} + (\bar{\boldsymbol{W}}_j^k)^T \sum_{l|A_{il}=1, i\neq l} \boldsymbol{H}_l^{(k-1)} = H_{ij}^k - S_{ij}^k \tag{50a}$$

$$H_{ij}^k \leq U_{ij}^k Z_{ij}^k \tag{50b}$$

$$S_{ij}^k \leq -L_{ij}^k (1 - Z_{ij}^k) \tag{50c}$$

There are two parts which constitute to the total bound. The first which is associated with the root node i, is calculated in a similar fashion as the MLP FBBT. The second part is calculated in the same way as the GCN, however, with the root node i ommited. This results in the following bound propagation equations.

$$\begin{aligned} U_{ij}^k = &\sum_{s=1}^{n_{k-1}} \max\left\{\hat{w}_{js}^k \max\left\{0, U_{sj}^{k-1}\right\}, \hat{w}_{js}^k \max\left\{0, L_{sj}^{k-1}\right\}\right\} \\ &+ \frac{\sqrt{d_{max}}}{\sqrt{2}} \sum_{s=1}^{n_{k-1}} \max\left\{\bar{w}_{js}^k \max\left\{0, U_{sj}^{k-1}\right\}, \bar{w}_{js}^k \max\left\{0, L_{sj}^{k-1}\right\}\right\}, \end{aligned} \tag{51a}$$

$$\begin{aligned} L_{ij}^k = &\sum_{s=1}^{n_{k-1}} \min\left\{\hat{w}_{js}^k \max\left\{0, U_{sj}^{k-1}\right\}, \hat{w}_{js}^k \max\left\{0, L_{sj}^{k-1}\right\}\right\} \\ &+ \frac{\sqrt{d_{max}}}{\sqrt{2}} \sum_{s=1}^{n_{k-1}} \min\left\{\bar{w}_{js}^k \max\left\{0, U_{sj}^{k-1}\right\}, \bar{w}_{js}^k \max\left\{0, L_{sj}^{k-1}\right\}\right\}. \end{aligned} \tag{51b}$$

Notice that $\frac{\sqrt{d_{max}}}{\sqrt{2}}$, has no plus one in the numerator underneath the square, because the root node is omitted.

For a similar formulation as Eqs. (44) and (45), the following set of equations can be considered:

$$U_{ij}^k = \max\left\{t_{ij}^k : t_{ij}^k \in C_{ij}^k\right\} \tag{52a}$$

$$L_{ij}^k = \min\left\{t_{ij}^k : t_{ij}^k \in C_{ij}^k\right\} \tag{52b}$$

$$C_{ij}^k = \left\{t_{ij}^k : t_{ij}^k = \hat{w}_j^k y^{k-1} + \frac{\sqrt{d_{max}}}{\sqrt{2}} \bar{w}_j^k x^{k-1}, x^{k-1}, y^{k-1} \in \left[\max\left\{0, L_i^{k-1}\right\}, \max\left\{0, U_i^{k-1}\right\}\right] \subset \mathbb{R}^{n_{k-1}}\right\} \tag{52c}$$

where $\hat{w}$ is the weight matrix associated with the root node, and $\bar{w}$ is the weight matrix multiplied with the aggregated nodes.

The upper bounds calculated in this section also serve as the values for the big-M constraints in (37d), (37e) and (37f). For instance, consider constraint (37d):

$$\boldsymbol{H}_l^k - \boldsymbol{M}(1 - A_{il}) \leq \boldsymbol{b}_{il}^k \tag{53}$$

and let there be no connection between node i and node l. In that case b_{il}^k needs to be able to be equal to 0. Enough $\boldsymbol{M}$ needs to be subtracted such that the left hand side of the equation is lower than 0. To achieve this $\boldsymbol{M}$ needs to be larger than H_l^k, which can be achieved if $\boldsymbol{M}$ is equal to the upper bound of H_l^k.

4.6 Genetic Algorithm for the optimisation of a trained GNN for molecular property prediction

The final section in this chapter concerns a genetic algorithm. As mentioned in Section 3.2.2, every GA contains the same steps. These steps include (1) the generation of the initial population, (2) the definition of the fitness

function, (3) a selection procedure, (4) the application of the crossover procedure, (5) the application of the mutation procedure, and (6) the definition of the stopping criterion. To our knowledge there exists no GA for the optimisation of trained graph neural networks. The following few paragraphs will describe the steps for the optimisation of a trained graph neural network. Specifically, we are optimising a network with 14 features and input constraints as those described in Section 4.4.1.

4.6.1 Initialisation, Fitness and Selection

To initialise an initial population, n_g graphs are generated all containing n nodes. To generate these graphs, first a random degree sequence S of length n is generated, where every entry S_i is a number randomly selected from the set $\{1, \dots, d_{\max}\}$. This degree sequence is then used define the edges between the nodes, such that the degree of the nodes matches the degree in the degree sequence. From here we can extract the adjacency matrix A.

Next the feature vectors must be initialised. We are trying to generate molecules which are also in the search space constrained by constraints (38). This means there are only single bonds and there are 14 features. The n_g matrices A, already contain information about the number of neighbours, thus defining $x_{i,5}, \dots x_{i,9}$. Features $x_{i,1}, \dots x_{i,4}$ and $x_{i,10}, \dots x_{i,14}$ still need to be generated. Note that there is an interaction between features $x_{i,1}, \dots x_{i,4}$ and $x_{i,10}, \dots x_{i,14}$, when the number of neighbours is known. If the atom type is known, the covalence is known, and when the number of neighbours is extracted we automatically know the number of hydrogen neighbours defined in $x_{i,10}, \dots x_{i,14}$. Therefore, only the atom types need to be generated.

The generation of the atom types for a graph is based on on the degree sequence S. The degree limits the possible atoms types of a node i. If the degree of a node is higher than the covalence associated with the atom type, this atom type cannot be assigned to a node. So for the generation of the atom type of a node i, first the degree S_i is assessed, and thereafter an atom type is chosen of which the covalence is higher than this degree S_i. This results in a atom type sequence T.

In the next steps we have to define a fitness function and selection procedure. The fitness function is defined by the GNNs, in the case of this thesis, this is either the GCN model and the GraphSAGE model. The only exception is that there is a heavy penalty for unconnected graphs. The selection procedure is a combination of roulette wheel selection procedure and elitism. The group which is not part of the molecules selected in the elites, will undergo the crossover and mutation procedure.

4.6.2 Crossover for GNNs for Chemical Property Modelling

Often in GAs, the crossover procedure is performed on a chromosome. We therefore convert the adjacency matrix of the generated graphs to a chromosome. The considered adjacency matrices are all symmetric, and are thus defined by the upper triangle of the matrix minus the diagonal. The entries of this upper diagonal, defined by $C_r = (A_{(r,r+1)}, \dots A_{(r,n)})^T$ for $r \in \{1, \dots, n-1\}$, are concatenated into a binary vector C. For instance, for the adjacency matrix A in figure 6, the adjacency matrix is converted to the binary vectors $C_1 = (0,1,1,0,0,0)^T, C_2 = (1,0,0,0,1)^T, C_3 = (0,0,0,0)^T, C_4 = (1,1,1)^T, C_5 = (1,1)^T, C_6 = (0)^T$, which are concatenated into vector $C = (0,1,1,0,0,0,1,0,0,0,1,0,0,0,0,1,1,1,1,1,0)^T$. On two of these vectors $C1$ and $C2$ the single point crossover procedure is performed.

The position of the single point crossover in C also defines the position of the crossover in the atom type

$$A = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

Figure 6: Adjacency matrix of a random molecule

sequence T. Let k be the position of the crossover in $C1$ and $C2$. The point k falls in one of the concatenated rows r of the vectors $C1$ and $C2$. That r defines the position in the cross over for T.

For example, let $k = 12$, then the crossover lies in C_3. This is because $|C_1| + |C_2| = 11 <= k <= 15 = |C_1| + |C_2| + |C_3|$. Then, let $T = (4, 1, 2, 1, 1, 1, 2)$ be an atom sequence associated with A (corresponding to atoms (Cl, C, O, C, C, C, O)). The single point crossover in the atom sequence thus occurs on position $r = 3$. Meaning $T_{\text{head}} = (4, 1, 2)$ and $T_{\text{tail}} = (1, 1, 1, 2)$. The crossover for the two atom sequences $T1$ and $T2$ associated with the binary vectors $O1$ and $O2$, result in the atom type offspring $OT1$ and $OT2$.

The offspring are thus two new adjacency matrix representing vectors $O1$ and $O2$ and two atom vectors $TO1$ and $TO2$. These matrix vectors and atom vectors get converted to feature vectors in a similar fashion as described in the paragraph on initial population generation.

There might be instances of an instance of offspring $(O1, OT1)$ which is not a feasible molecule. To make the molecule feasible, we go through the following steps:

1. While the amount of connected components is more than 1, we add edges from one connected component to nodes outside of that connected component. Then we check whether there are nodes with a degree higher than $d_{\max}$ and remove edges connected to that node.
2. It can also be that the string results in an adjacency matrix which is connected, but still some nodes have a degree higher than $d_{\max}$. In this case we also remove edges from that node until the degree is lower or equal than $d_{\max}$. There is a small possibility that this causes an undirected graph, however, unconnected graphs gets heavily penalised in the fitness function.
3. After step 1 and 2, there is a high possibility that the graphs that are found are connected, and they certainly have a degree of maximum $d_{\max}$. We check what the allowed degree is based on atom type sequence $OT1$ and associated covalence of the atoms in that sequence. We compare this with the degree of the nodes in the graph. If the degree is higher than the allowed degree, we mutate the molecules into a random molecule which has the covalence which allows for the degree of the node.

After these steps the resulting molecule is congruent with the constraints of (38).

4.6.3 Mutation and Terminating the Algorithm

For the mutation step, with a probability P_m, a random entry of the offspring O, representing the new adjacency matrix, gets selected, and is flipped. The resulting molecule might be infeasible. In that case the same procedure is applied as in the crossover procedure until the molecule is feasible. After the string is mutated, the atom types are also mutated. With a probability of P_{ma} an atom in the sequence OT gets switched to another molecule. Once again, the fixing procedure is applied as described in the cross over procedure section.

Finally, the algorithm terminates after τ seconds.

4.7 Summary

To summarise, we have introduced a few novel things in this chapter. We have introduced an MINLP formulation of the GCN model and a MILP formulation of the GraphSAGE model. We have introduced methods to constrain the input space of an MI(N)LP formulation such that molecule-like solutions can be found. This input space formulation is modelled using the input of GNNs, which are an adjacency matrix and feature vectors. Finally, we introduce a new way to run a GA to optimise trained GNNs, which works with adjacency matrices and feature vectors in string representation. This is different from previous attempts where the GNN uses a latent space, on which the GA is applied.

When comparing the MILP formulation of the GCN and the MINLP formulation of the GraphSAGE network, the first thing we notice is that the MINLP formulation is bi-linear, whereas the MILP formulation is linear. Linear solvers are known to be faster than non-linear solvers. Another drawback for the GCN formulation is that initially, a vast amount of constraints and variable are introduced to linearise the normalisation term. This is not the case for the GraphSAGE model.

However, the amount of constraints and variables introduced in the GraphSAGE formulation is far greater than for the GCN model for deep and wide networks. This is because for every layer k the amount of constraints for the GraphSAGE model grows by $\mathcal{O}(n^2 n_k)$ whereas the GCN model only grows by $\mathcal{O}(n^2)$. For the variables we also find that the increase is of the order $\mathcal{O}(n^2 n_k)$ for GraphSAGE and $\mathcal{O}(n^2)$ for the GCN.

In the next chapter we will put the proposed theory into practice and see how the formulations perform when implemented into a deterministic solver.

5 Numerical Results

Recall that the goal of this thesis is to linearise graph neural networks such that they can be used as surrogate models in optimisation problems. To validate the MI(N)LP formulations, there is a focus on chemical property prediction, specifically the prediction of boiling points. In this chapter, the experiments will be described which will aid us in answering the research questions. Thereafter, the results of these experiments are presented.

5.1 Experimental Setup

The workflow of the experiments is as follows. First a data set is chosen, which will be used to train a neural network. Thereafter, this trained neural network is linearised using the methods described in Chapter 4. The linear formulations are optimised using a deterministic solver. These steps for both the initial experiments and the case study are described in more detail in this section.

The experiments were ran on two different machines, a laptop and a virtual machine. The laptop was used for quick, low resource intensive experiments in which we were only interested in the MI(N)LP solutions. The laptop was used as the implementation of experiments on this machine were quick and simple. The virtual machine was used for experiments where solving times were compared. These experiments took longer and required a constant CPU availability.

The laptop was a MacBook Pro (13-inch, 2020) with a 1,4 GHz Quad-Core Intel Core i5 processor, 8 GB 2133MHz LPDDR3 Ram and an Intel Iris Plus Graphics 645 1536 MB graphics card, running macOS Monterey. The virtual machine was a virtual linux machine with eight 2,5 Ghz Intel(R) Xeon(R) Gold 6248 CPUs, and 16GB RAM, running Ubuntu 20.04.5 LTS. The machine learning models were trained using PyTorch 1.11.0 [99], and implemented using PyTorchGeometric 2.0.4 [100]. The optimisation software used to find solutions for the optimisation problems was Gurobi version 9.5.1..

5.1.1 Initial Experiments

In this section we explain how the initial experiments were performed. The purpose of these initial experiments is to understand how the optimisation software performs in terms of solving time for different GNN configurations. With these experiments we would like to understand how the solving times (and optimality gaps) are impacted when the node width and layer depth are altered for both the GCN and GraphSAGE model. As a result, we get a better understanding of how the models perform and we can perform a comparative study between the GCN and GraphSAGE models.

The original data set that was used was one consisting of 192 molecular components, mostly refrigerants. Every compound in the data set was labelled with a boiling point T_b. The boiling points in the dataset ranged from 145.15 to 482.05 K. The atom types in the original data set are carbon (C), oxygen (O), fluorine (F), chlorine (Cl), bromine (Br), nitrogen (N) and sulphur (S). Early testing indicated that solving times of the MILP formulation increased with more features in the feature vectors. Therefore, the data set was analysed and atom types which were not frequently represented in the dataset (< 11 times) were removed. The resulting data set has 177 molecules, with a T_b range of $145.15 - 482.05$. The atom types that were included in the model are carbon (C), oxygen (O), fluorine (F) and chlorine (Cl).

The machine learning model was trained on the data set described above. The two graph neural networks that were trained are the Graph Convolutional Network by Kipf and Welling [7] and the GraphSAGE model

by Hamilton et al. [8]. All molecules were converted to a spatial representation, represented by a graph. Every atom in the graph is represented by a node, and the bonds between the atoms are represented by edges. Every node in the graph also had an associated feature vector, including descriptors, which represented information about that particular atom. The descriptors that were used in both configurations were 4 different atom types (carbon (C), oxygen (O), fluorine (F), chlorine (Cl)), the number of neighbours of an atom ($0-4$ neighbours) and how many hydrogen atoms were connected ($0-4$) atoms. All these descriptors were converted to a one-hot encoding, resulting in 14 features to each feature vector.

The hyper-parameters for both the GCN and GraphSAGE models were the same. The hyper-parameters were the same as in [96]. We used the same hyper-parameters since the training of the GNNs in this thesis was a simplified version of their GNN model. Those hyper-parameters were the following. The model quality was measured with the MSE. For training, all molecule boiling points were normalised. The number of epochs were 300, with an early stopping patience of 50. The learning rate was set to 0.001, where after 3 consecutive epochs without model improvement, the learning rate was decreased by 0.8.

There were 7 different configurations for the two GNNs. All GNNs had an input layer of 14 features per node, hidden layers and 32 two neurons for the output layer. The hidden layers differed in the number of layers and the node width, in the following configurations:

		node width		
		16	32	64
hidden layers	0	0		
	1	1×16	1×32	1×64
	2	2×16	2×32	2×64

All 7 GNN configurations were followed by an add pooling layer, forming a graph fingerprint. This fingerprint of 32 nodes was fed through a 3 layer MLP ($32 \rightarrow 16 \rightarrow 1$). All GNN configurations were trained 20 times with the above hyper-parameter settings on the laptop. The 20 trained networks were compared based on the validation data. The models with the lowest validation MSE were selected and used as parameters for the MILP formulation in the solver.

The complete formulations as described in Appendix A.1 and Appendix A.2 were implemented in Gurobi. The weights and biases were imported from the trained GNNs with the lowest validation MSE. The only parameter that still needed to be defined in the MILP input space constraints was the molecule length. The MILP formulations were solved to optimality with a molecule length of $4, 6$ and 8, resulting in 21 experiments for both GNNs. The experiments were ran for 10 hours on the virtual machine. The experiments were compared on the solving time and optimality gap. Also the objective values and the best found solutions were recorded.

All configurations, for both GNNs, were compared with a baseline. In this baseline, all formulations were optimised using a GA instead of the (non-)linear solver. The GA was implemented exactly as described in Section 4.6. The GA was initialised with 50 molecules. The GA was terminated after 36000 seconds, or if the GA found the best known objective value which was found using the deterministic optimiser. In the latter case the amount of seconds to find this solution was noted. The number of elites in each iterations was set to 0, because having a higher number of elites resulted in premature convergence. The crossover position was randomised for each pair in the formulation. The mutation probability of flipping the string bits was set to

$P_m = \frac{4}{|T|}$, where T is the bit string. The mutation probability of changing the atom types P_{ma} was set to $\frac{1}{n}$, where n is the length of the molecule.

5.1.2 Case Study

To review the output results of the proposed methods we performed a case study. The example shows a test case where a chemist models a chemical property and tries to optimise it. This is a common use case of CAMD [101]. The chemist can then use the found solutions and test the predicted properties instead of having to search over the entire search space of feasible molecules.

The input space constraints were extended by including the possibility to find double and triple bonds, implemented using the input constraints described by Eqs. (41). Initial testing of the experiments found problems with steric constraints on the model, and to circumvent this, molecules with loops were excluded from the search space.

The case study included the optimisation of the simplest GraphSAGE configuration with at least one hidden layer, which was the `1 × 16` configuration. A total of three different formulations were tested. The first where the search space included only single and double bonded molecules, the second single and triple bonded molecules, and the third included single, double and triple bonded molecules. There were 15, 15, and 16 input features respectively to account for the input space demands. For this reason, three different neural neural network configurations were trained. The rest of the neural network structure was exactly the same as the previously described experiment. The neural network was trained with the same hyper-parameters as the initial experiments and thus also trained for 20 iterations on the laptop. The network with the lowest validation error was selected as input for the MILP formulation. The formulation was solved to optimality on the laptop. Multiple solutions (max 8) were recorded for molecule lengths 4 and 5. All solutions were analysed, checking whether the found solutions were molecules which naturally exist in nature.

5.2 Results

The following section describes the results that were found in the experiments. First, the results are discussed for the GCN and GraphSAGE models individually. The results per model differ in node depth, layer width and molecule length. Thereafter, a comparison is described between models with the same parameters. Thereafter, the case study is described, showing which molecules were found using the proposed methods.

5.2.1 Initial Experiments

The first step in optimising a trained graph neural network is training the network that will be optimised. For both models, the 7 configurations (differing in the amount of hidden layers and nodes per hidden layer) were trained using the MSE as a loss function. A comparison of the box plots of the MSE of the models can be found in Fig. 7.

For the GCN model, the minimum MSE values range from 0.021182 to 0.027783 and the median MSE range from 0.035990 to 0.049458. For the GraphSAGE model the minimum MSE values range from 0.017135 to 0.027699 and the median from 0.030848 to 0.043802. In case of the GraphSAGE we can see that median MSE values decrease as the amount of nodes per layer increase. For GCN no such pattern can be detected.

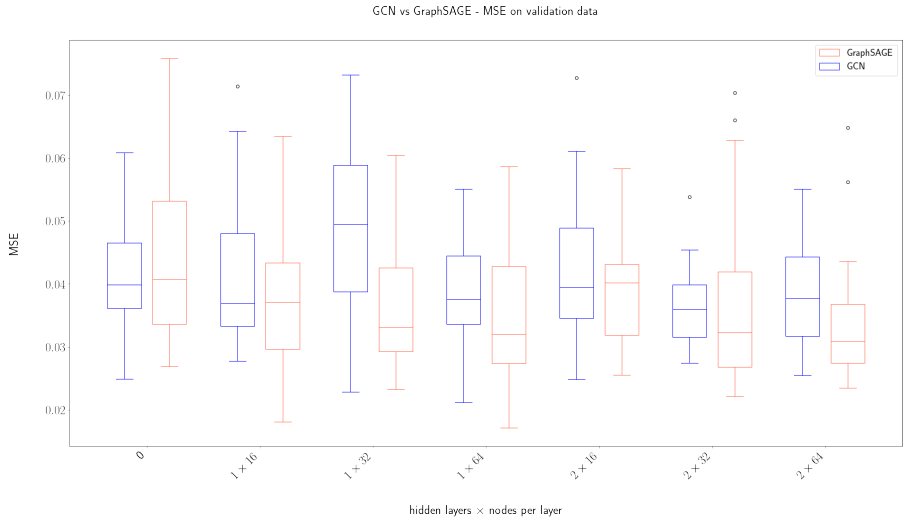


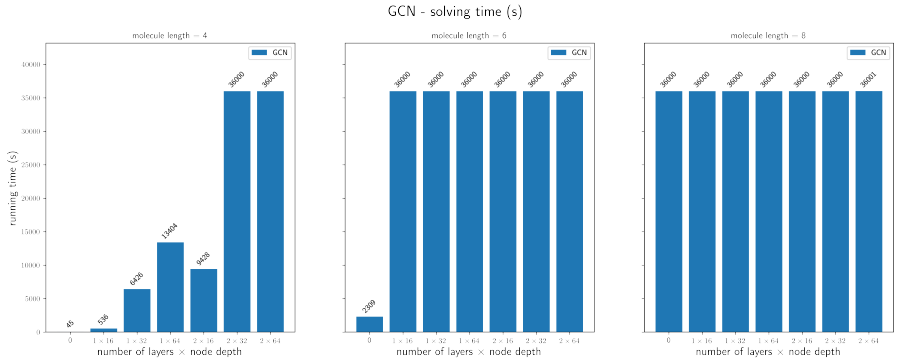
Figure 7: Box-plots of the MSE, for the GCN and GraphSAGE models for different layer depths and node width, after independently running the models 20 times for each configuration. The box-plots indicate the median, the lower and upper quartile and the lowest and largest MSE, when outliers, which are indicated by the dots, are excluded.

Comparing the model accuracy of both GNN models show that the GraphSAGE model has a better median MSE validation value for 4 out of the 7 configurations. The minimum MSE values also show the GraphSAGE model to be better for 4 out of the 7 configurations. The best overall minimum MSE is found in the `1 x 64` configuration of the GraphSAGE model, with an MSE of 0.017135. However, the `1 x 16` is a close second with an MSE of 0.018115.

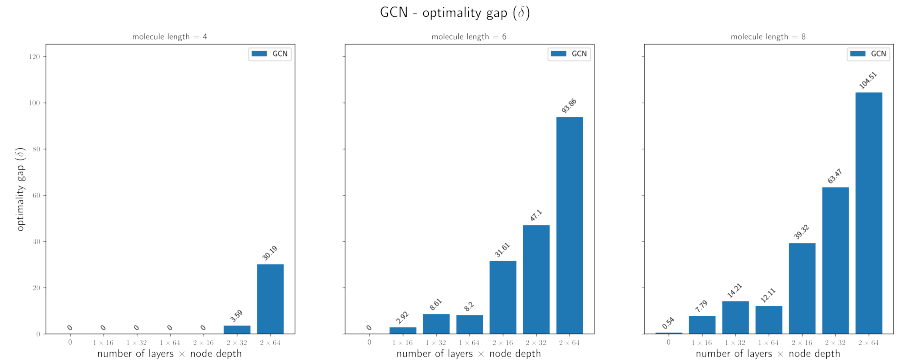
Using the trained models as parameters for the MILP formulation, we optimised the MILP and MINLP formulations of the GNNs for 10 hours. In Fig 8 the results for the optimisation of the MINLP formulation are compared for the different configurations. First we consider the solving times. Six out of the 21 experiments were able to find an optimal solution before the preset stopping time of 10 hours. The other 15 experiments were terminated after 10 hours. All configuration were solved to optimality in $n = 4$ apart from `2 x 32` and `2 x 64`. For molecules length $n = 6$ only the formulation with 0 hidden layers was solved to optimality. For $n = 4$, we can see that the as the amount of nodes per layer increase, the solving time increases. For $n = 4$, we can also say that as the layer depth increases, the solving times also increases.

The optimality gaps are non-zero for the configurations where an optimum is not found. Recall that the optimality gap indicates how far the upper bound is removed from the lower bound, expressed in multiples of the lower bound (See Eq. (4)). For $n = 4$ the optimality gaps range from 0 to 30.19, for $n = 6$ they range from 0 to 93.86 and for $n = 8$ they range from from 0.54 to 104.51. As the node depth increases, the optimality gap increases, apart from increasing the node depth from 32 to 64 with 1 hidden layer for $n = 6$ and $n = 8$. As the layer depth increases, the optimality gap also increases for all non-solved configurations. Finally, we note that all optimality gaps increase as the molecule length increases.

Fig 9 shows the results of optimising the MILP formulations of the trained GraphSAGE neural networks. The solving time results show that 5 configurations were solved to optimality. All the others terminated after a time limit of 10 hours. Of the solved cases, four optima where found when the search space was limited to atoms of length 4 ($n = 4$), the other one was found when $n = 6$. Once again, when node depth increases, the



(a) The solving time in seconds of the GCN MINLP formulation while optimising for different molecule lengths (smaller is better)



(b) The optimality gap of the GCN MINLP formulation while optimising for different molecule lengths (smaller is better)

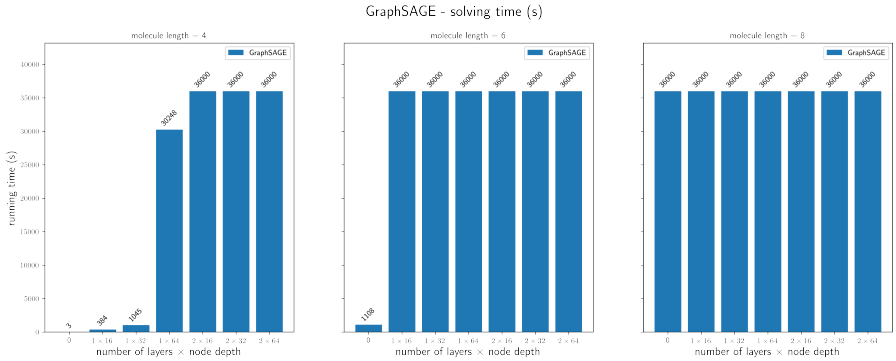
Figure 8: Solving times and optimality gaps for the GCN MINLP formulation after 36000 seconds of optimisation. The experiments differ in number of hidden layers and the node depth, and the amount of atoms per molecule.

solving times increase. The same goes for when the number of hidden layers increase, the solving time increases. Where the solving time does not reach the limit of 10 hours, the solving time also increases when the length of the molecules increase.

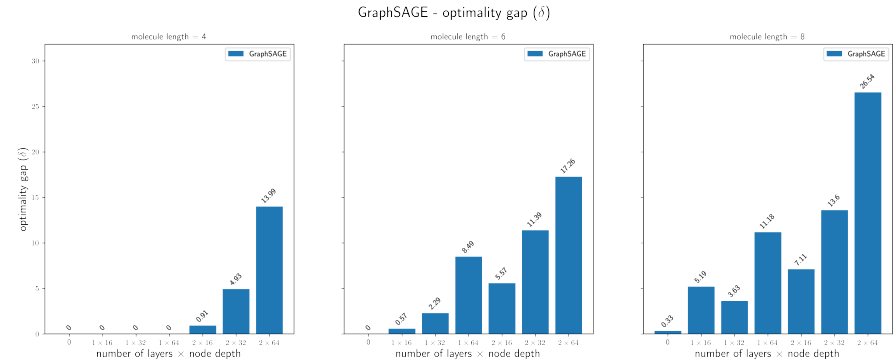
In Fig 9(b), the results are shown for the optimality gap of the MILP formulation of the GraphSAGE network. For $n = 4$ the optimality gaps range from 0 to 13.99, for $n = 6$ they range from 0 to 17.62 and for $n = 8$ they range from from 0.33 to 26.54. We see that for $n = 4, 6$, as the node depth increases, the optimality gaps become larger. For $n = 8$, this is not the case. When increasing the node depth for one hidden layer from 16 to 32, the optimality gap decreases. When increasing the number of layers, the optimality gap always gets larger when the node depth stays the same, for all molecule lengths. Finally as the molecule length increase, the optimality gap also increases.

Next we compare the GCN and GraphSAGE formulations based on solving time and the optimality gap. The comparison of the optimality gap can be found in Fig 10. As mentioned before, the GCN model solves to optimality 6 times, whereas the GraphSAGE model solves to optimality 5 times. When solved to optimality, the GraphSAGE model is generally faster, apart from the configurations `1 x 64` and `2 x 16` for $n = 4$, where in the latter case GCN solves to optimality and GraphSAGE does not.

Figure 11 shows the comparative results of the optimality gap for both GNN formulations. We see that

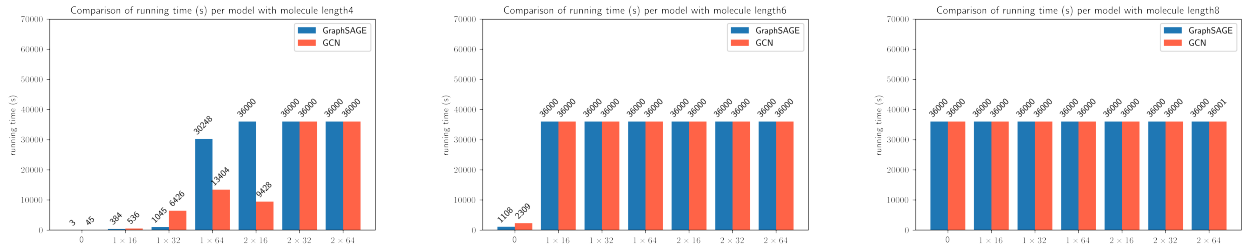


(a) The solving time in seconds of the GraphSAGE MILP formulation while optimising for different molecule lengths (smaller is better)



(b) The optimality gap of the GraphSAGE MILP formulation while optimising for different molecule lengths (smaller is better)

Figure 9: From friends to graphs

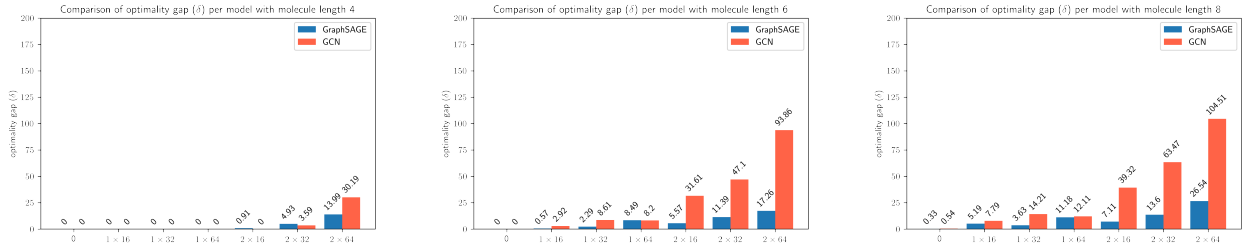


(a) solving time per model, n = 4

(b) solving time per model, n = 6

(c) solving time per model, n = 8

Figure 10: solving times (s)



(a) optimality gap per model, n = 4

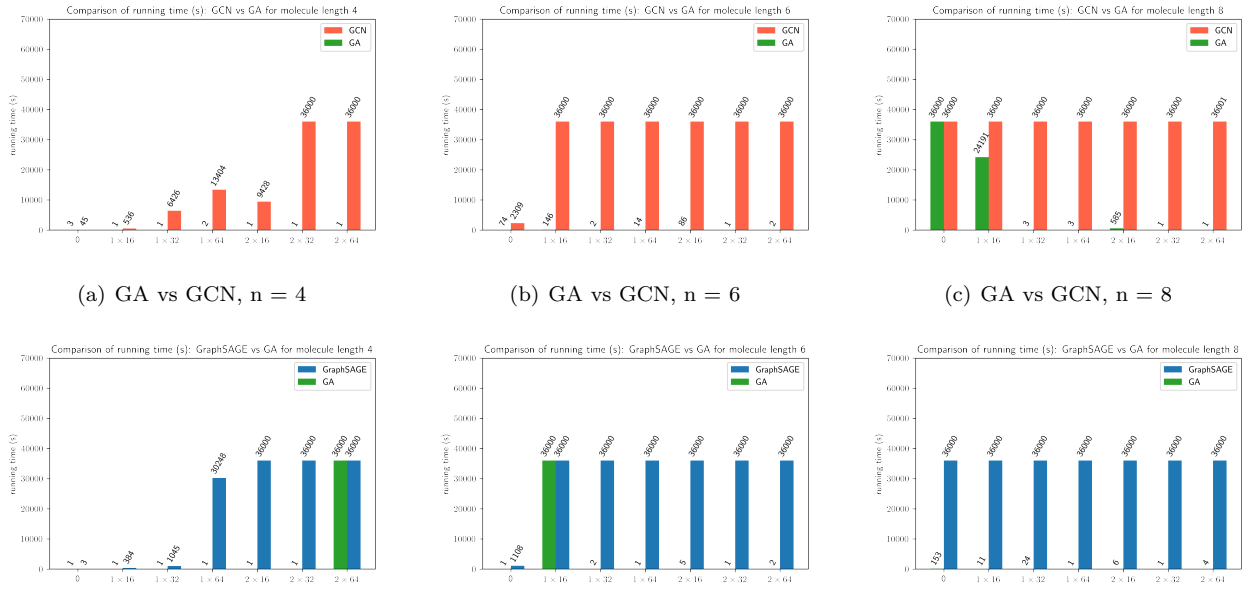
(b) optimality gap per model, n = 6

(c) optimality gap per model, n = 8

Figure 11: optimality gaps

GraphSAGE has a better optimality gap than the GCN formulation for most configurations and molecule lengths. There are a few exceptions however. For $n = 4$, `2 x 16` and `2 x 32`, GCN has a smaller optimality gap than the GraphSAGE formulation. The same goes for $n = 6$ with 1 hidden layer and 64 nodes in that layer.

Finally we compare the solving times of the GNNs with å baseline. The plots in Fig 12 show this comparison. For 35 out of 42 instances (83%), the GA found an equally good solution as the deterministic optimiser in less than 2 minutes. For 3 instances, the GA did not find an equally good objective value and were terminated because it reached the time limit of 10 hours.



(a) GA vs GCN, n = 4

(b) GA vs GCN, n = 6

(c) GA vs GCN, n = 8

(d) GA vs GraphSAGE, n = 4

(e) GA vs GraphSAGE, n = 6

(f) GA vs GraphSAGE, n = 8

Figure 12: Comparison of the GA and GNN. The solving time in seconds for the GA indicates after how many seconds the GA found an objective value of equal quality or better than the MILP formulation of the GNN.

5.2.2 Case Study

For the case study we selected the GraphSAGE GNN with 1 hidden layer and 16 nodes. The reason for choosing this model is explained in Appendix B.1. As explained in Section 5.1.2, there were three different search spaces for the molecules. These were the search space of single and double bonded molecules, single and triple bonded molecules, and single, double and triple bonded molecules. Graphical representations of the found solutions for $n = 4$ and $n = 5$ can be found in Fig. 13.

There are repeat molecules in the solution set. These are solutions which have different adjacency matrices but constitute to the same molecule. In total there were 20 unique molecule like structures found during the

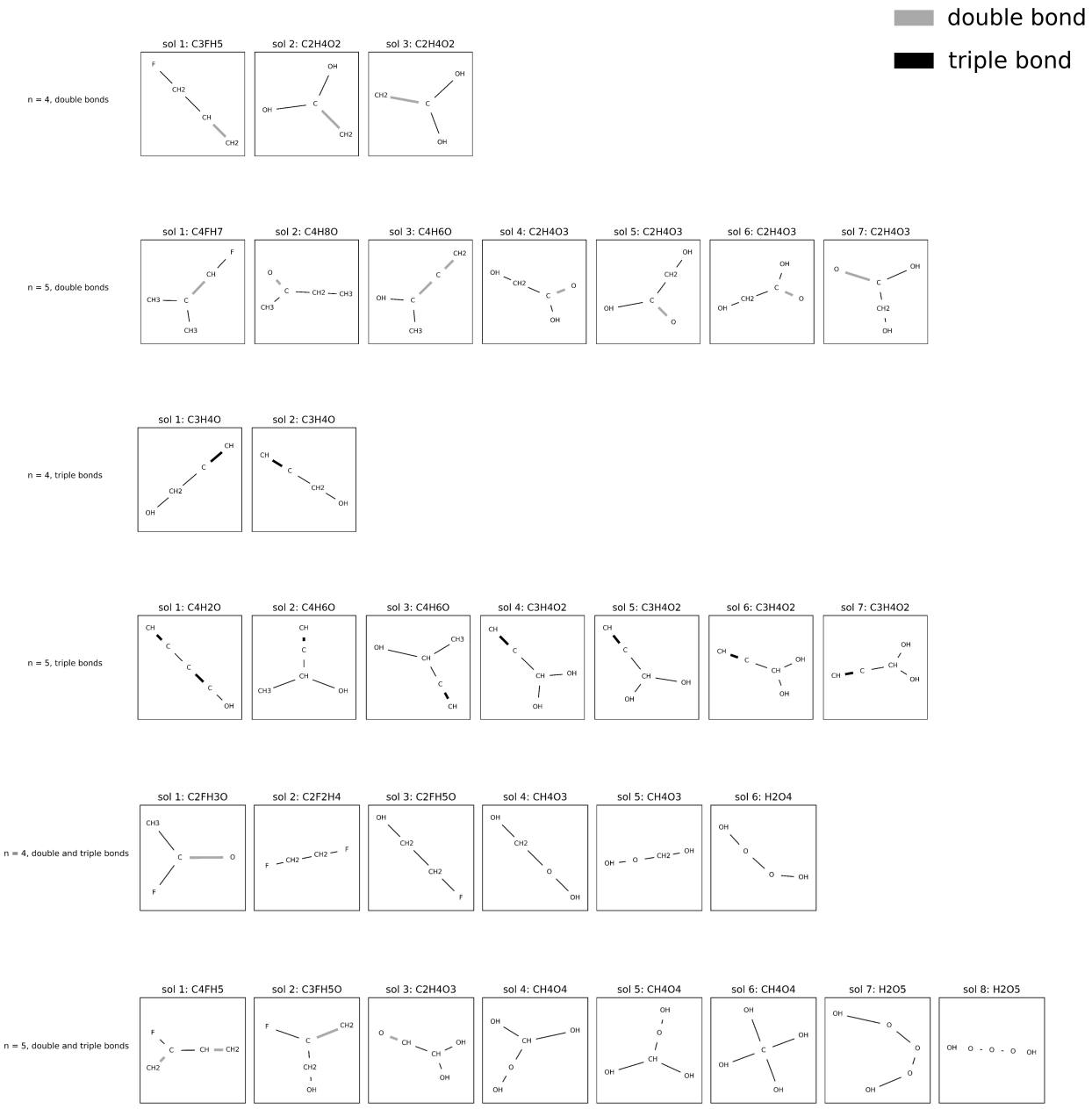


Figure 13: The molecules associated with the best found lower bounds during the branch and bound optimisation process of the GraphSAGE formulation with 1 hidden layer and 16 nodes.

optimisation of the different search spaces. The analysis of these molecules can be found in Tab. 3.

For 12 of the 20 molecules, we found sources indicating that the molecules where experimentally observed. For the 12 observed molecules, 9 where synthesised and their boiling points where recorded. Two of the molecules were not experimentally observed but where mentioned in papers as hypothetically possible under large pressure. 2 of the molecules found during the optimisation process where also present in the training data set, and all the others were not.

The absolute difference between the experimental observed boiling points and the predicted boiling points from the optimisation ranged from 9.44 to 72.24. The relative difference, calculated by the difference divided by the experimental boiling point, ranged from $2.7\% - 24.2\%$.

bonds	molecule length	formula	experimentally observed	experimental T_b (K)	molecular name	In training dataset?	objective value	predicted T_b (K)	difference
double bonded	4	C3FH5	TRUE *	253.15	Allyl fluoride	FALSE	0.02	313.17	60.02
		C2H4O2	TRUE [102]		1,1-Dihydroxyethene	FALSE	0.78	361.61	
	5	C4FH7	FALSE		n/a	FALSE	0.27	329.79	
		C4H8O	TRUE **	343.15	butanone	TRUE	0.63	352.59	9.44
		C4H6O	FALSE		n/a	FALSE	0.72	357.90	
		C2H4O3	TRUE **	375.15	Glycolic acid	FALSE	1.35	397.92	22.77
triple bonded	4	C3H4O	TRUE **	377.15	Propargyl alcohol	FALSE	0.57	348.49	-28.66
	5	C4H2O	TRUE [103]		butadiynol	FALSE	0.19	324.31	
		C4H6O	TRUE *	329.15	(±)-3-Butyn-2-ol	FALSE	0.82	364.57	35.42
		C3H4O2	FALSE		2-Propyne-1,1-diol	FALSE	1.12	383.40	
double and triple bonded	4	C2FH3O	TRUE ***	283.15	Acetyl fluoride	FALSE	0.20	324.94	41.79
		C2F2H4	TRUE *	293.85	1,2-Difluorethane	TRUE	0.71	357.30	63.45
		C2FH5O	TRUE **	366.15	2-Fluoroethanol	FALSE	1.15	385.06	18.91
		CH4O3	FALSE		Hydroperoxymethanol	FALSE	1.63	415.34	
		H2O4	TRUE [104]		Tetraoxidane	FALSE	1.80	425.95	
	5	C4FH5	FALSE		2-Fluoro-1,3-butadiene	FALSE	0.01	313.17	
		C3FH5O	TRUE ***	298.15	2-Fluoro-2-propen-1-ol	FALSE	0.92	370.39	72.24
		C2H4O3	FALSE		Dihydroxyacetaldehyde	FALSE	1.49	406.47	
		CH4O4	FALSE [105] +		Orthocarbonic acid	FALSE	2.33	459.23	
		H2O5	FALSE [104] +		Pentaoxidane	FALSE	2.39	463.40	

Table 3: Table with all experimental results and interpretation of the case study. Experimentally observed instances indicated with a star come from a database of a chemical supplier where * = Matrix Scientific, ** = Alfa Aesar, *** = Synquest. + = hypothetical molecule in cited paper. The predicted $T_b(K)$ is calculated by $T_b(K) = \text{mean} + \text{std} \times \text{obj\_val}$, mean = 312.64, std = 62.98, where the mean and std are from the training data set.

6 Conclusion and Outlook

In this final chapter we first discuss whether or not we can confirm the research hypotheses. We then discuss any interesting results regarding the initial experiments which were not discussed in the research questions. This section is followed by an interpretation of the case study. The strengths and weaknesses of the used methodology are discussed and is then related to other work in the literature. We conclude with a section introducing ideas for further research.

6.1 Discussions of the Research Questions

We start with answering the research questions:

1. *Can we formulate Graph Convolutional Networks (GCN) as Mixed Integer (Non-) Linear Problems?*

The goal of this thesis was to formulate trained graph neural networks as MILP programming formulations, such that they could be used as surrogate models in optimisation problems. A frequently-used model was the GCN by Kipf and Welling [7], and was thus chosen to be formulated such that it could be optimised by a solver. We can safely conclude by the theory in Section 4.2 and the results in Section 5.2.1, that we succeeded in formulating the GCN as an MINLP.

2. *Can we achieve similar model accuracy with the GraphSAGE GNN of which the architecture extends more naturally to MILP formulations resulting in decreased solving times?*

There are two parts to answering this question. First, we want to answer whether we can get similar GNN accuracy between the GCN model and the GraphSAGE model. Second, we want to answer whether the MILP formulation of the GraphSAGE model solves faster than the MINLP formulation of the GCN.

First, we theorised that the GCN would get better model accuracy with less hidden layers and nodes per layer than GraphSAGE due to the GCN model's more complex architecture. The results as plotted in Fig 7, do not support this hypothesis. For 4 out of the 7 configurations (hidden layers $\times$ nodes) the GraphSAGE model has a better median validation MSE while running for 20 iterations, and for 4 out of 7 configurations, the GraphSAGE model has a lower minimum validation error than the GCN model. To conclude, with our hyper-parameters as described in Section 5.1, we achieved similar model accuracy for both the GCN and GraphSAGE model, even with the same number of hidden layers and nodes per layer.

To answer the second part of this question, the comparison of the solving times of the two models are considered, as laid out in Fig. 10. Our hypothesis was that the GraphSAGE MILP formulations would be faster than the GCN MINLP formulations because linear solvers are faster than non-linear solvers. As mentioned in the result section, the GraphSAGE model was generally faster (4 out of 6 solved instances). The optimality gaps also seem to suggest that if the experiments were ran for longer the GraphSAGE would generally solve to optimality first. This is because for all but 3 configurations (12 out of 15 early terminated cases) the GraphSAGE model had a smaller optimality gap than the GCN model. However, we should note that having a smaller optimality gap does not immediately mean that the would be solving times are going to be faster. Generally this is true however, because there are more possibilities to prune by bound.

Overall, there evidence to suggest the MILP formulation of the GraphSAGE model solves to optimality faster than the MINLP formulation of the GCN model, with similar model accuracy. This is because our trained model accuracy is about the same and sometimes better for the GraphSAGE model compared to the GCN model, for

models with similar hidden layers and number of nodes, combined with the fact that the GraphSAGE model often solves to optimality faster with similar configurations.

The reason for the unexpected instances, where the GCN model solved to optimality faster than the GraphSAGE model, can be due to the properties of the trained GNNs. We will go over these ideas in Section 6.2.

6.2 A discussion of the results of the experiments

In the following section the results of the experiments will be discussed which were not discussed when answering the research questions. The section starts with a general discussion of the results of optimising GNN models. We then discuss the comparison of the two models. Finally we discuss the case study.

6.2.1 Individual Experiments

First of all, we go into the results of the initial experiments. First we note that for very few instances the solver actually solves to optimality, for both the GCN model and the GraphSAGE model. This means that, in its current configuration, the proposed formulations can only be used to find small graphs of size 4 in a time span of 10 hours. There are improvements that can be made to improve solving times, which will be discussed later. However, even with improvements we do not expect the search space to be able to include graphs which are multiple orders of magnitude larger than the graphs that we currently find. This means that the proposed optimisation formulations can not be used in other contexts where large graph neural networks are used, like road network modelling, or recommender systems. This implies that that the proposed techniques, in its current formulation, should be used for small graph optimisations only, like molecule optimisation.

Next, we note that the use of the a deterministic optimiser has the advantage of knowing how close one is to the actual solution, while running the algorithm, expressed by the optimality gap. However, the experiments show that optimality gaps rapidly increase as the model becomes more complex, or as the search space includes larger graphs. When modelling some instances, this optimality gap might not be as useful anymore. For instance, in the case that $n = 8$, the optimality gap was 26.54 after running the `2 x 64` instance of the GraphSAGE model for 10 hours. The range of boiling points in the training set ranged from $145.15 - 482.05$ K. With the found objective lower bound, the optimality gap of 26.54 implies that the solution lies in a range of approximately $675 - 2045$ K. For the set of refrigerants we could assume beforehand that the temperatures were in this boiling point range for molecules of length 8.

Next we discuss the simple fact that, when the amount of nodes per layer increase, the solving time goes up for both the GCN and GraphSAGE. The same pattern can be seen when increasing the hidden layers per model. These results are as expected for general mixed integer linear programming formulations. As the amount of layers and nodes increase, the amount of decision variables and constraints increase, making the problem more difficult to solve. The optimality gap shows similar results apart from a few exceptions as laid out in the result description section. The cases where unexpected results were seen were rerun, but resulted in similar optimality gaps, implying that the problem lay somewhere with the learned parameters of the GNN. We further explore this in the next section.

Finally, a general remark on the bounds for the GNNs. The input bound size for the MLP which comes after the pooling layer increase linearly with the amount of nodes that are in the graphs in the search space. This is because the output bounds of the feature vectors of the GNNs get summed in the pooling layer. In

some cases this makes sense. Larger structures sometimes result in higher objective values, as is the case with boiling points of molecules. However, when bound are loose to start with, it amplifies this error, resulting in even larger bounds. This has a negative impact on the solving times.

6.2.2 Discussion of the Comparison of the Experiments

As discussed when answering the second research question, the GraphSAGE model is generally better than the GCN model in terms of solving times, and when not solved to optimality, also in terms of an optimality gap. The general conclusion should thus be to use the GraphSAGE model when training the model, in case a researcher is indifferent on the model type.

There are instances where the GCN is better than the GraphSAGE model. First we note that the bounds for the GCN models are smaller than the GraphSAGE model, for equal node depth and layer width. As mentioned before, smaller bounds result in faster solving times. However, the bound difference is generally present for all instances when comparing the GCN and GraphSAGE. There thus must be another reason for these exceptions.

Training a neural network is a stochastic process. This means that training different neural networks with the same hyper-parameters does not result in the same weights and biases. Our hypothesis is that training different trained graph neural networks with the same configurations result in different solving times. Having different weights and biases has an impact on the bounds. In turn, we know that larger bounds have a negative impact on the solving time. We tested this hypothesis as can be seen in Appendix B.2. The same experiment for the instance `2 x 16` was repeated 5 times. It shows that different trained neural network parameters result in different solving times when optimised using a deterministic solver. This confirms our hypothesis. However, we find no correlation between the bounds and the solving time. We expected larger bounds to result in slower solving times, but this small test in the appendix does not confirm this hypothesis.

The final point to discuss with regards to the initial experiments are the genetic algorithm baseline comparisons. It is clear from the results that in most cases the genetic algorithm is superior to the deterministic solvers in terms of finding a solution of equal quality, while taking less time.

There are three instances where the GA does not find a solution of equal quality. In these cases the GA gets stuck in a local maximum. These instances also illustrate the shortcomings of the GA as an optimisation method. The deterministic solvers, as the name suggest, always find the best objective value, when given enough time, albeit taking exponentially more time when graph sizes increase. For the GA, the researcher is left in the dark as to the quality of the found solution. It is up to the needs of the researchers to decide which method is appropriate for the goal they are trying to achieve.

6.2.3 Discussion of the Case Study

The goal of the case study was to emulate an instance where a researcher is looking for molecules with a maximal boiling point. There are a few interesting things that can be discussed regarding the found molecules.

First of all, 12 of the molecules that were found were experimentally observed. Of the other 8, we able to find two which were mentioned in research as hypothetical molecules. These were able to be synthesised under very high pressure or were an unstable molecule of molecular reaction. Of the other 6, we were unable to find any mentions in literature.

We also note that only two of the 20 molecules that were found were in the original data set. We believe

that this shows that a model can be trained on a particular data set, and that other molecules can be found outside of that data set, of which some can be synthesised. This means there is a real life use case for the proposed formulation in this thesis. Let us say a researcher wants to design a fuel with high energy storage, and low emissions. With GNNs the researcher is able to model these chemical properties. Using the methods proposed in this thesis the researcher can make an MILP formulation of these networks, and find solutions while optimising. The researcher can use these solution as a starting point to look for molecules with the desired properties, instead of having to start with a pool of all possible fuels.

There are two final remarks we would like to make on the found solutions. Again, these should be taken with a grain of salt, as the modelling of the chemical properties was not the main focus of this thesis. However, we do see that when experimental results exist of molecules with similar input constraints and molecule length, the experimental boiling points increase as the modelled boiling points increase. This shows some validation for the modelling quality. On the other hand, we note that the mean absolute error of the trained GNN is about 6.65. For the found molecules, of which experimental boiling point data is available, we see that our mean absolute error is around 17.75. Without further exploration we can not draw immediate conclusions from this. However, one hypothesis is that this might suggest that when modelled molecules are at the higher end of the boiling point spectrum, that the errors of the GNNs become larger.

6.2.4 Shortcomings

There are a few shortcomings in the research that was conducted. First of all, it should be noted that training of the GNNs was not the main focus of this thesis. The goal was to show that trained GNNs could be formulated as MILPs such that they could be used in surrogate models. The resulting trained neural network models are thus not a very good representation of the learned molecules. This mostly impacted the case study as the found molecules had quite large errors compared to the experimental boiling point data available. Therefore the case study should be seen more as an inspiration of what can be achieved, more so than a real focus on the found results.

There are a few ways in which the results of training the neural networks could be improved. First of all, the size of the training data set could be increased. There is a lot of experimental data available on different molecules and this can decrease the validation error on the trained neural networks. Next, we used a fraction of the learnable features. Examples of these features are the hybridisation, ring structure, aromatic structure, etc. The reason was because we were not able to implement these features as constraints in the linear input space. There might be features which can possibly be linearised, and thus improve the model. This can improve the neural network quality in further research.

A shortcoming in the testing of our MI(N)LP formulations is that we tested all formulations just once. This means that we were only testing the solving time for a specific trained neural network. This might have a negative impact on the results as we are not only comparing the formulations, but there is another aspect that has impact on the solving time which we have no control over. A better option would be to run all 42 experiments multiple times, and then take average solving time and optimality gap. However, since the experiments took 10 hours per experiment this was not possible to do in the allotted time for this thesis.

6.2.5 Related Research

The significance of the research in relationship to established knowledge is twofold. First of all, the research extends the neural network architectures which can be deterministically optimised. It therefore stands as an extension of the early work of Fischetti and Jo [11] and Tjeng et al. [12]. As far as we know, it is thus far the only other attempt of a linearisation of a neural network architecture other than the feedfoward MLP.

Second, we were only able to find one other research paper where the optimisation of trained graph neural networks was used in CAMD. This is in a preprint by Rittig et al. [10]. In this publication the trained graph neural networks are optimised using Bayesian optimisation and a genetic algorithm. Both of these instances are not deterministic optimisation techniques. The significance of our research in relation to this research paper, lies in the fact that a specific class of GNNs, for small molecule lengths, can be deterministically optimised.

A small side note in relation to the paper by Rittig et al. [10] is that their optimisation using the GA uses a GNN architecture which uses a latent space. The GA is applied on this latent space. The GA in this thesis does not use a latent space formulation.

6.2.6 Future Research

For future research there are quite a few different directions one can take. First if we consider the formulation itself, the most obvious improvement which can be made to this model are better bound tightening techniques. As mentioned before the bounds for the formulations in this thesis were really loose, due to using the feasibility based bound tightening techniques. Other bound tightening techniques like optimisation based bound tightening techniques (OBBT) could improve the bounds found for this model. Examples of these are the optimisation based techniques introduced by Fischetti and Jo [11] or by Tjeng et al. [12]. But also the bound tightening technique introduced by Wang et al. [65] which is a combination of feasibility and optimality based bound tightening would be promising. We see no reason why the bound tightening techniques of regular MLP MILP formulations can not be adapted to work for GNN MILP formulations as well. We expect decreased solving times, making it possible to find larger graph structures, or increase the model complexity.

Since the MILP formulations of these GNNs can be used as surrogate models, there is also a possibility that there are output bounds. Grimstad and Andersson [17] introduced bound tightening techniques which propagate output bounds backward. For GNNs, bounds after the pooling layer are often very large when searching for graphs with a large number of nodes as is explained in Section 6.2.1. We therefore believe that a backward propagating FBBT can already have significant impact on the solving times, while it is computationally cheap to implement.

Using the GNNs in CAMD also pose some interesting research directions. First of all, an interesting research direction is taking a look at how to improve the model accuracy of modelling molecules while still using GNN structures which are linearisable. From our conclusions, we noted that using linearisable GNN models is preferable, especially on larger graph structures. An example of a direction which can be explored is the k-GNN architecture introduced by Schweidtmann et al. [96], in combination with the GraphSAGE model, and omitting the edge weight network. The research shows promising model quality and we believe that it is linearisable.

Next we consider the input constraints. We note that a lot of logical operators can be encoded in linear programming formulations as constraints. Examples of these are the logical `AND`, `OR`, `XOR` and `AND`. An interesting point of research would be to see whether it is theoretically possible to model the search space to include only

real synthesizable molecules, while only using logical operators and other linear programming constraints. This would form the basis to assess whether optimising MILPs of trained GNNs is a useful research direction in CAMD. If it can be proven that this is not possible, it would be interesting to research which subset of molecules can be modelled using linear constraints only. In that case, a researcher interested in CAMD using the proposed techniques understands for which research problems the techniques can be applied.

When deterministic optimisation is not the main priority for researchers we believe that using GAs for optimising trained graph neural networks is the best option. Although introducing the GA was not the main priority of this research, and thus the GA could be greatly improved, the results of even this GA were very promising. We showed similar solution quality with greatly reduced speeds. Advantages of MILP formulations, like showing other good solutions can also be included in the programming of the GA.

Thereby, the GA has the possibility to be more versatile than linear programming. First of all, any graph neural network architecture can be used, including models with edge weights. This can be done by simply changing the fitness function. This allows a molecular designer to model the molecules better. Another advantage of using a GA is that the molecule generation can be non-linear. Solutions that are generated with the GA can be discounted for instance when steric constraints are violated. Due to its versatility and speed we think that optimising trained GNNs with GAs is an exiting opportunity for CAMD. Although research already exists where octane fuels have been designed [10], many more avenues in CAMD can be explored using GAs to optimise GNNs.

References

[1] B Firdaus Begam and J Satheesh Kumar. Computer assisted qsar/qspr approaches–a review. *Indian J Sci Technol*, 9(8):1–8, 2016.

[2] Fabiana Alves de Lima Ribeiro and Márcia Miguel Castro Ferreira. Qspr models of boiling point, octanol–water partition coefficient and retention time index of polycyclic aromatic hydrocarbons. *Journal of Molecular Structure: THEOCHEM*, 663(1-3):109–126, 2003.

[3] SH Hilal, SW Karickhoff, and LA Carreira. Prediction of the vapor pressure boiling point, heat of vaporization and diffusion coefficient of organic compounds. *QSAR & Combinatorial Science*, 22(6): 565–574, 2003.

[4] Nick D Austin, Nikolaos V Sahinidis, and Daniel W Trahan. Computer-aided molecular design: An introduction and review of tools, applications, and solution techniques. *Chemical Engineering Research and Design*, 116:2–26, 2016.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[6] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32 (1):4–24, 2020.

[7] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[8] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[9] Oliver Wieder, Stefan Kohlbacher, Mélaine Kuenemann, Arthur Garon, Pierre Ducrot, Thomas Seidel, and Thierry Langer. A compact review of molecular property prediction with graph neural networks. *Drug Discovery Today: Technologies*, 37:1–12, 2020.

[10] Jan G Rittig, Martin Ritzert, Artur M Schweidtmann, Stefanie Winkler, Jana M Weber, Philipp Morsch, K Alexander Heufer, Martin Grohe, Alexander Mitsos, and Manuel Dahmen. Graph machine learning for design of high-octane fuels. *arXiv preprint arXiv:2206.00619*, 2022.

[11] Matteo Fischetti and Jason Jo. Deep neural networks and mixed integer linear optimization. *Constraints*, 23(3):296–309, 2018.

[12] Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017.

[13] Rudy Bunel, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and M. Pawan Kumar. Piecewise linear neural network verification: A comparative study. *CoRR*, abs/1711.00455, 2017. URL http://arxiv.org/abs/1711.00455.

[14] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output range analysis for deep neural networks. *CoRR*, abs/1709.09130, 2017. URL http://arxiv.org/abs/1709.09130.

[15] Abhinav Kumar, Thiago Serra, and Srikumar Ramalingam. Equivalent and approximate transformations of deep neural networks. *arXiv preprint arXiv:1905.11428*, 2019.

[16] Thiago Serra, Abhinav Kumar, and Srikumar Ramalingam. Lossless compression of deep neural networks. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 417–430. Springer, 2020.

[17] Bjarne Grimstad and Henrik Andersson. Relu networks as surrogate models in mixed-integer linear programs. *Computers & Chemical Engineering*, 131:106580, 2019.

[18] Marcello Di Martino, Styliani Avraamidou, and Efstratios N Pistikopoulos. A neural network based superstructure optimization approach to reverse osmosis desalination plants. *Membranes*, 12(2):199, 2022.

[19] Alyssa Kody, Samuel Chevalier, Spyros Chatzivasileiadis, and Daniel Molzahn. Modeling the ac power flow equations with optimally compact neural networks: Application to unit commitment. *Electric Power Systems Research*, 213:108282, 2022.

[20] Shu-Bo Yang, Zukui Li, and Wei Wu. Data-driven process optimization considering surrogate model prediction uncertainty: A mixture density network-based approach. *Industrial & Engineering Chemistry Research*, 60(5):2206–2222, 2021.

[21] Jan G Rittig, Qinghe Gao, Manuel Dahmen, Alexander Mitsos, and Artur M Schweidtmann. Graph neural networks for the prediction of molecular structure-property relationships. *arXiv preprint arXiv:2208.04852*, 2022.

[22] Alan R Katritzky, Victor S Lobanov, and Mati Karelson. Qspr: the correlation and quantitative prediction of chemical and physical properties from structure. *Chemical Society Reviews*, 24(4):279–287, 1995.

[23] Zhanyao Ha, Zbigniew Ring, and Shijie Liu. Quantitative structure- property relationship (qspr) models for boiling points, specific gravities, and refraction indices of hydrocarbons. *Energy & fuels*, 19(1):152–163, 2005.

[24] Matthew D Wessel and Peter C Jurs. Prediction of normal boiling points of hydrocarbons from molecular structure. *Journal of chemical information and computer sciences*, 35(1):68–76, 1995.

[25] Leanne M Egolf, Matthew D Wessel, and Peter C Jurs. Prediction of boiling points and critical temperatures of industrially important organic compounds from molecular structure. *Journal of Chemical Information and Computer Sciences*, 34(4):947–956, 1994.

[26] Ovidiu Ivanciuc, Teodora Ivanciuc, and Alexandru T Balaban. Quantitative structure–property relationships for the normal boiling temperatures of acyclic carbonyl compounds. *Internet Electron. J. Mol. Des*, 1:252–268, 2002.

[27] Mohamed Roubehie Fissa, Yasmina Lahiouel, Latifa Khaouane, and Salah Hanini. Qspr estimation models of normal boiling point and relative liquid density of pure hydrocarbons using mlr and mlp-ann

methods. *Journal of Molecular Graphics and Modelling*, 87:109–120, 2019. ISSN 1093-3263. doi: https://doi.org/10.1016/j.jmgm.2018.11.013. URL https://www.sciencedirect.com/science/article/pii/S1093326318306673.

[28] Yi-min Dai, Zhi-ping Zhu, Zhong Cao, Yue-fei Zhang, Ju-lan Zeng, and Xun Li. Prediction of boiling points of organic compounds by qspr tools. *Journal of Molecular Graphics and Modelling*, 44:113–119, 2013.

[29] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[30] Johann Gasteiger and Jure Zupan. Neural networks in chemistry. *Angewandte Chemie International Edition in English*, 32(4):503–527, 1993.

[31] Igor I Baskin, David Winkler, and Igor V Tetko. A renaissance of neural networks in drug discovery. *Expert opinion on drug discovery*, 11(8):785–795, 2016.

[32] David M Himmelblau. Applications of artificial neural networks in chemical engineering. *Korean journal of chemical engineering*, 17(4):373–392, 2000.

[33] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE international joint conference on neural networks*, volume 2, pages 729–734, 2005.

[34] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[35] Claudio Gallicchio and Alessio Micheli. Graph echo state networks. In *The 2010 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2010.

[36] Alessandro Lusci, Gianluca Pollastri, and Pierre Baldi. Deep architectures and deep learning in chemoinformatics: the prediction of aqueous solubility for drug-like molecules. *Journal of chemical information and modeling*, 53(7):1563–1575, 2013.

[37] Hiroyuki Shindo and Yuji Matsumoto. Gated graph recursive neural networks for molecular property prediction. *arXiv preprint arXiv:1909.00259*, 2019.

[38] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, 31(7):1235–1270, 2019.

[39] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[40] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.

[41] Alessio Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009. doi: 10.1109/TNN.2008.2010350.

[42] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023. PMLR, 2016.

[43] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1416–1424, 2018.

[44] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[45] Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. *arXiv preprint arXiv:1803.07294*, 2018.

[46] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5115–5124, 2017.

[47] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[48] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[49] Elman Mansimov, Omar Mahmood, Seokho Kang, and Kyunghyun Cho. Molecular geometry prediction using a deep generative graph neural network. *Scientific reports*, 9(1):1–13, 2019.

[50] Michael Withnall, Edvard Lindelöf, Ola Engkvist, and Hongming Chen. Building attention and edge message passing neural networks for bioactivity and physical–chemical property prediction. *Journal of cheminformatics*, 12(1):1–18, 2020.

[51] Han Altae-Tran, Bharath Ramsundar, Aneesh S Pappu, and Vijay Pande. Low data drug discovery with one-shot learning. *ACS central science*, 3(4):283–293, 2017.

[52] Giorgos Bouritsas, Fabrizio Frasca, Stefanos P Zafeiriou, and Michael Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[53] Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard S Zemel. Lanczosnet: Multi-scale deep graph convolutional networks. *arXiv preprint arXiv:1901.01484*, 2019.

[54] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[55] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. *Advances in neural information processing systems*, 28, 2015.

[56] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. *arXiv preprint arXiv:1912.09893*, 2019.

[57] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

[58] Dmitry B Kireev. Chemnet: a novel neural network based method for graph/property mapping. *Journal of chemical information and computer sciences*, 35(2):175–180, 1995.

[59] Kevin Yang, Kyle Swanson, Wengong Jin, Connor Coley, Philipp Eiden, Hua Gao, Angel Guzman-Perez, Timothy Hopper, Brian Kelley, Miriam Mathea, et al. Analyzing learned molecular representations for property prediction. *Journal of chemical information and modeling*, 59(8):3370–3388, 2019.

[60] Chi Chen, Weike Ye, Yunxing Zuo, Chen Zheng, and Shyue Ping Ong. Graph networks as a universal machine learning framework for molecules and crystals. *Chemistry of Materials*, 31(9):3564–3572, 2019.

[61] Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and counting linear regions of deep neural networks. In *International Conference on Machine Learning*, pages 4558–4566. PMLR, 2018.

[62] Juan Pablo Vielma. Mixed integer linear programming formulation techniques. *Siam Review*, 57(1):3–57, 2015.

[63] Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. Maximum resilience of artificial neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pages 251–268. Springer, 2017.

[64] Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja, and Juan Pablo Vielma. Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, 183 (1):3–39, 2020.

[65] Keliang Wang, Leonardo Lozano, Carlos Cardonha, and David Bergman. Acceleration techniques for optimization over trained neural network ensembles. *arXiv preprint arXiv:2112.07007*, 2021.

[66] Amit P Duvedi and Luke EK Achenie. Designing environmentally safe refrigerants using mathematical programming. *Chemical Engineering Science*, 51(15):3727–3739, 1996.

[67] Nachiket Churi and Luke EK Achenie. Novel mathematical programming model for computer aided molecular design. *Industrial & engineering chemistry research*, 35(10):3788–3794, 1996.

[68] Manish Sinha, Luke EK Achenie, and Gennadi M Ostrovsky. Environmentally benign solvent design by global optimization. *Computers & Chemical Engineering*, 23(10):1381–1394, 1999.

[69] Nikolaos V Sahinidis, Mohit Tawarmalani, and Minrui Yu. Design of alternative refrigerants via global optimization. *AIChE Journal*, 49(7):1761–1775, 2003.

[70] Venkat Venkatasubramanian, King Chan, and James M Caruthers. Computer-aided molecular design using genetic algorithms. *Computers & Chemical Engineering*, 18(9):833–844, 1994.

[71] Braam Van Dyk and Izak Nieuwoudt. Design of solvents for extractive distillation. *Industrial & engineering chemistry research*, 39(5):1423–1429, 2000.

[72] Weiyu Xu and Urimila M Diwekar. Improved genetic algorithms for deterministic optimization and optimization under uncertainty. part ii. solvent selection under uncertainty. *Industrial & engineering chemistry research*, 44(18):7138–7146, 2005.

[73] Robert H Herring III and Mario R Eden. Evolutionary algorithm for de novo molecular design with multi-dimensional constraints. *Computers & Chemical Engineering*, 83:267–277, 2015.

[74] Jan Scheffczyk, Lorenz Fleitmann, Annett Schwarz, Matthias Lampe, André Bardow, and Kai Leonhard. Cosmo-camd: A framework for optimization-based computer-aided molecular design using cosmo-rs. *Chemical Engineering Science*, 159:84–92, 2017.

[75] Bao Lin, Sunitha Chavali, K Camarda, and David C Miller. Computer-aided molecular design using tabu search. *Computers & Chemical Engineering*, 29(2):337–347, 2005.

[76] Sunitha Chavali, Bao Lin, David C Miller, and Kyle V Camarda. Environmentally-benign transition metal catalyst design using optimization techniques. *Computers & chemical engineering*, 28(5):605–611, 2004.

[77] Francesca Grisoni, Michael Moret, Robin Lingwood, and Gisbert Schneider. Bidirectional molecule generation with recurrent neural networks. *Journal of chemical information and modeling*, 60(3):1175–1183, 2020.

[78] Esben Jannik Bjerrum and Richard Threlfall. Molecular generation with recurrent neural networks (rnns). *arXiv preprint arXiv:1705.04612*, 2017.

[79] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *International conference on artificial neural networks*, pages 412–422. Springer, 2018.

[80] Qi Liu, Miltiadis Allamanis, Marc Brockschmidt, and Alexander Gaunt. Constrained graph variational autoencoders for molecule design. *Advances in neural information processing systems*, 31, 2018.

[81] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *International conference on machine learning*, pages 2323–2332. PMLR, 2018.

[82] Artur Kadurin, Sergey Nikolenko, Kuzma Khrabrov, Alex Aliper, and Alex Zhavoronkov. drugan: an advanced generative adversarial autoencoder model for de novo generation of new molecules with desired molecular properties in silico. *Molecular pharmaceutics*, 14(9):3098–3104, 2017.

[83] Artur Kadurin, Alexander Aliper, Andrey Kazennov, Polina Mamoshina, Quentin Vanhaelen, Kuzma Khrabrov, and Alex Zhavoronkov. The cornucopia of meaningful leads: Applying deep adversarial autoencoders for new molecule development in oncology. *Oncotarget*, 8(7):10883, 2017.

[84] Oleksii Prykhodko, Simon Viet Johansson, Panagiotis-Christos Kotsias, Josep Arús-Pous, Esben Jannik Bjerrum, Ola Engkvist, and Hongming Chen. A de novo molecular generation method using latent vector based generative adversarial network. *Journal of Cheminformatics*, 11(1):1–13, 2019.

[85] Gabriel Lima Guimaraes, Benjamin Sanchez-Lengeling, Carlos Outeiral, Pedro Luis Cunha Farias, and Alán Aspuru-Guzik. Objective-reinforced generative adversarial networks (organ) for sequence generation models. *arXiv preprint arXiv:1705.10843*, 2017.

[86] Benjamin Sanchez-Lengeling, Carlos Outeiral, Gabriel L Guimaraes, and Alan Aspuru-Guzik. Optimizing distributions over molecular space. an objective-reinforced generative adversarial network for inverse-design chemistry (organic). 2017.

[87] Zhenpeng Zhou, Steven Kearnes, Li Li, Richard N Zare, and Patrick Riley. Optimization of molecules via deep reinforcement learning. *Scientific reports*, 9(1):1–10, 2019.

[88] Marcus Olivecrona, Thomas Blaschke, Ola Engkvist, and Hongming Chen. Molecular de-novo design through deep reinforcement learning. *Journal of cheminformatics*, 9(1):1–14, 2017.

[89] Mark Newman. *Networks*. Oxford university press, 2018.

[90] Laurence A Wolsey. *Integer programming*. John Wiley & Sons, 2020.

[91] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, 1984.

[92] George B Dantzig. Application of the simplex method to a transportation problem. *Activity analysis and production and allocation*, 1951.

[93] Michele Conforti, Gérard Cornuéjols, Giacomo Zambelli, et al. *Integer programming*, volume 271. Springer, 2014.

[94] Samuel Burer and Adam N Letchford. Non-convex mixed-integer nonlinear programming: A survey. *Surveys in Operations Research and Management Science*, 17(2):97–106, 2012.

[95] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, 80(5):8091–8126, 2021.

[96] Artur M Schweidtmann, Jan G Rittig, Andrea Konig, Martin Grohe, Alexander Mitsos, and Manuel Dahmen. Graph neural networks for prediction of fuel ignition quality. *Energy & fuels*, 34(9):11395–11407, 2020.

[97] Lei Zhang, Stefano Cignitti, and Rafiqul Gani. Generic mathematical programming formulation and solution for computer-aided molecular design. *Computers & Chemical Engineering*, 78:79–84, 2015.

[98] Ambros M Gleixner, Timo Berthold, Benjamin Müller, and Stefan Weltge. Three enhancements for optimization-based bound tightening. *Journal of Global Optimization*, 67(4):731–757, 2017.

[99]

[100] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[101] Horst Frühbeis, Robert Klein, and Holger Wallmeier. Computer-assisted molecular design (camd)—an overview. *Angewandte Chemie International Edition in English*, 26(5):403–418, 1987.

[102] Artur Mardyukov, André K Eckhardt, and Peter R Schreiner. 1, 1-ethenediol: The long elusive enol of acetic acid. *Angewandte Chemie International Edition*, 59(14):5577–5580, 2020.

[103] Mitsunori Araki and Nobuhiko Kuze. Laboratory detection of a linear carbon chain alcohol: Hc4oh and its deuterated species. *The Astrophysical Journal*, 680(1):L93, 2008.

[104] Alexander V Levanov, Dmitri V Sakharov, Anna V Dashkova, Ewald E Antipenko, and Valeri V Lunin. Synthesis of hydrogen polyoxides h2o4 and h2o3 and their characterization by raman spectroscopy, 2011.

[105] Stanislav Böhm, Diana Antipova, and Josef Kuthan. A study of methanetetraol dehydration to carbonic acid. *International journal of quantum chemistry*, 62(3):315–322, 1997.

A MI(N)LP formulations of the GNNs

A.1 GCN

$$\boldsymbol{W}^{K_{MLP}}\boldsymbol{x}^{(K_{MLP}-1)} + b^K = x_1^K \tag{54a}$$

$$\boldsymbol{W}_j^k\boldsymbol{x}^{k-1} + b_j^k = x_j^k - s_j^k \qquad \forall k \in \{1,\dots,K_{MLP}-1\}, \forall j \in \{1,\dots,n_k\} \tag{54b}$$

$$x_j^k \leq U_j^k z_j^k \qquad \forall k \in \{1,\dots,K_{MLP}-1\}, \forall j \in \{1,\dots,n_k\} \tag{54c}$$

$$s_j^k \leq -L_j^k(1-z_j^k) \qquad \forall k \in \{1,\dots,K_{MLP}-1\}, \forall j \in \{1,\dots,n_k\} \tag{54d}$$

$$x_j^0 = \sum_i H_{ij}^K \qquad j \in \{1,\dots,n_K\} \tag{54e}$$

$$\sum_l \hat{s}_{il}\boldsymbol{W}_j^{kT}\boldsymbol{H}_l^{(k-1)} = H_{ij}^k - S_{ij}^k \qquad \forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{54f}$$

$$H_{ij}^k \leq U_{ij}^k Z_{ij}^k \qquad \forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{54g}$$

$$S_{ij}^k \leq -L_{ij}^k(1-Z_{ij}^k) \qquad \forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{54h}$$

$$d_i^+ = \sum_j \tilde{A}_{ij} \qquad \forall i \in \{1,\dots,N\} \tag{54i}$$

$$p_{il} = d_i^+(d_{max}+1) + d_l^+ \qquad \forall i,l \in \{1,\dots,N\} \tag{54j}$$

$$0 = p_{il} - 1c_1^{il} - 2c_2^{il} - \cdots - (d_{max}+1)^2 c_{(d_{max}+1)^2}^{il} \qquad \forall i,l \in \{1,\dots,N\} \tag{54k}$$

$$1 = c_1^{il} + \cdots + c_{(d_{max}+1)^2}^{il} \qquad \forall i,l \in \{1,\dots,N\} \tag{54l}$$

$$s_{il} = c_1^{il}g_1 + \cdots + c_{(d_{max}+1)^2}^{il} g_{(d_{max}+1)^2} \qquad \forall i,l \in \{1,\dots,N\} \tag{54m}$$

$$s_{il} - M(1-A_{il}) \leq \hat{s}_{il} \leq M(1-A_{il}) + s_{il} \qquad \forall i,l \in \{1,\dots,N\} \tag{54n}$$

$$-MA_{il} \leq \hat{s}_{il} \leq MA_{il} \qquad \forall i,l \in \{1,\dots,N\} \tag{54o}$$

$$0 \leq x_j^k, s_j^k \in \mathbb{R} \qquad \forall k \in \{1,\dots,K_{MLP}\}, \forall j \in \{1,\dots,n_k\} \tag{54p}$$

$$z_j^k \in \{0,1\} \qquad \forall k \in \{1,\dots,K_{MLP}-1\}, \forall j \in \{1,\dots,n_k\} \tag{54q}$$

$$0 \leq H_{ij}^k, S_{ij}^k \in \mathbb{R} \qquad \forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{54r}$$

$$Z_{ij}^k \in \{0,1\} \qquad \forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{54s}$$

$$\hat{s}_{il}, s_{il} \in \mathbb{R} \qquad \forall i,l \in \{1,\dots,N\} \tag{54t}$$

$$p^{il}, c^{il} \in \{0,1\} \qquad \forall i,l \in \{1,\dots,d_{max}+1\} \tag{54u}$$

$$d_i^+ \in \{0,1\} \qquad \forall i \in \{1,\dots,d_{max}+1\} \tag{54v}$$

$$A_{ij} \in \{0,1\} \qquad \forall i,j \in \{1,\dots,N\} \tag{54w}$$

$$H^0, A \in \Omega \tag{54x}$$

A.2 GraphSAGE

$$\boldsymbol{W}^{K_{MLP}}\boldsymbol{x}^{(K_{MLP}-1)} + b^K = x_1^{K_{MLP}} \tag{55a}$$

$$\boldsymbol{W}_j^k\boldsymbol{x}^{k-1} + b_j^k = x_j^k - s_j^k \qquad \forall k \in \{1,\dots,K_{MLP}-1\}, \forall j \in \{1,\dots,n_k\} \tag{55b}$$

$$x_j^k \leq U_j^k z_j^k \qquad \forall k \in \{1,\dots,K_{MLP}-1\}, \forall j \in \{1,\dots,n_k\} \tag{55c}$$

$$s_j^k \leq -L_j^k(1-z_j^k) \qquad \forall k \in \{1,\dots,K_{MLP}-1\}, \forall j \in \{1,\dots,n_k\} \tag{55d}$$

$$x_j^0 = \sum_i H_{ij}^K \qquad j \in \{1,\dots,n_K\} \tag{55e}$$

$$(\hat{\boldsymbol{W}}_j^k)^T\boldsymbol{H}_i^{(k-1)} + (\bar{\boldsymbol{W}}_j^k)^T\sum_l \boldsymbol{b}_{il}^{(k-1)} = H_{ij}^k - S_{ij}^k \quad \forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{55f}$$

$$H_{ij}^k \leq U_{ij}^k Z_{ij}^k \qquad \forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{55g}$$

$$S_{ij}^k \leq -L_{ij}^k(1-Z_{ij}^k) \qquad \forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{55h}$$

$$\boldsymbol{H}_l^k - \boldsymbol{M}(1-A_{il}) \leq \boldsymbol{b}_{il}^k \qquad \forall k \in \{0,\dots,K-1\}, \forall i,l \in \{1,\dots,N\} \tag{55i}$$

$$\boldsymbol{b}_{il}^k \leq \boldsymbol{H}_l^k + \boldsymbol{M}(1-A_{il}) \qquad \forall k \in \{0,\dots,K-1\}, \forall i,l \in \{1,\dots,N\} \tag{55j}$$

$$-\boldsymbol{M}(A_{il}) \leq \boldsymbol{b}_{il}^k \leq \boldsymbol{M}(A_{il}) \qquad \forall k \in \{0,\dots,K-1\}, \forall i,l \in \{1,\dots,N\} \tag{55k}$$

$$0 \leq x_j^k, s_j^k \in \mathbb{R} \qquad \forall k \in \{1,\dots,K_{MLP}\}, \forall j \in \{1,\dots,n_k\} \tag{55l}$$

$$z_j^k \in \{0,1\} \qquad \forall k \in \{1,\dots,K_{MLP}-1\}, \forall j \in \{1,\dots,n_k\} \tag{55m}$$

$$0 \leq H_{ij}^k, S_{ij}^k \in \mathbb{R} \qquad \forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{55n}$$

$$Z_{ij}^k \in \{0,1\} \qquad \forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{55o}$$

$$\boldsymbol{b}_{il}^k \in \mathbb{R}^{n_k} \qquad \forall k \in \{1,\dots,K\}, \forall i,l \in \{1,\dots,N\} \tag{55p}$$

$$A_{ij} \in \{0,1\} \qquad \forall i,j \in \{1,\dots,N\} \tag{55q}$$

$$H^0, A \in \Omega \tag{55r}$$

B Extra results

B.1 Deciding on the Model for the Case Study

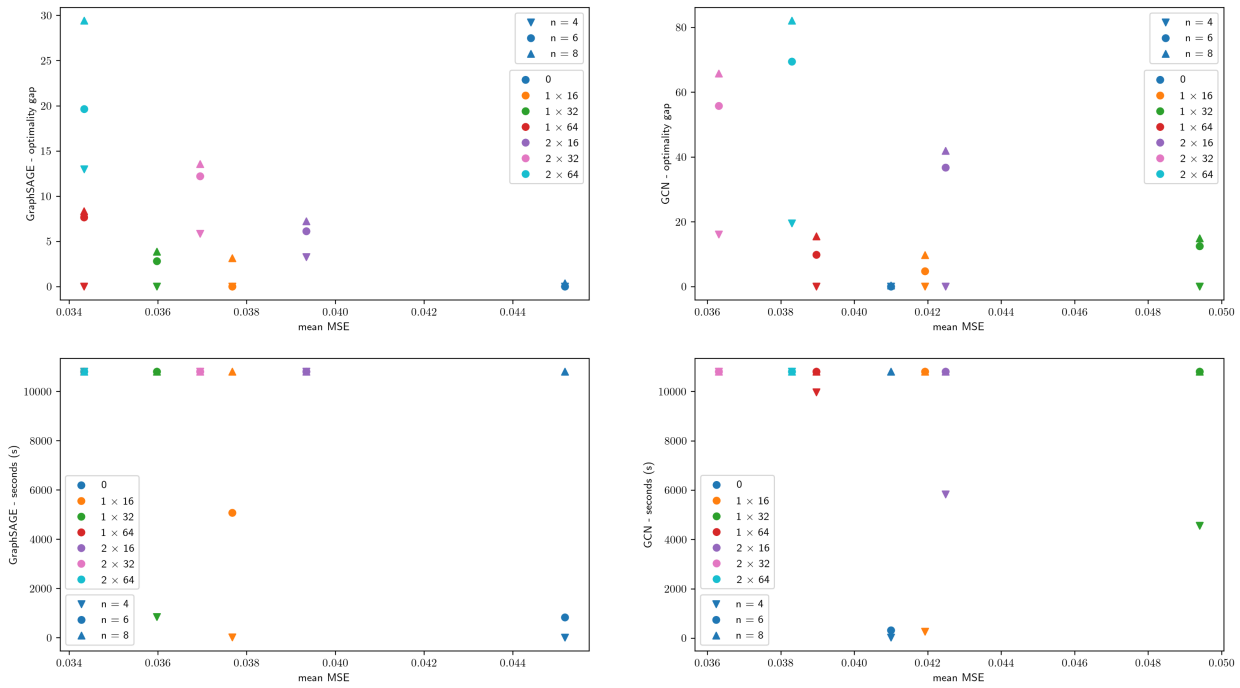


Figure 14: Caption

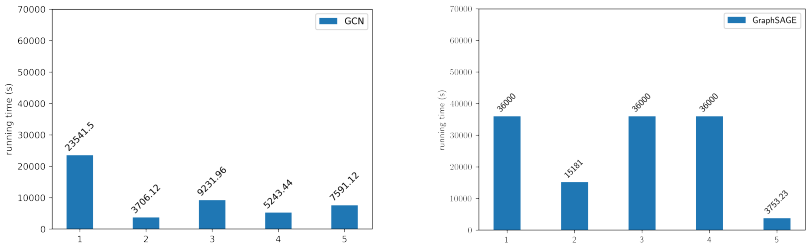
Fig. 14 was used to decide which formulation to use for the case study. Since from our other results we noted that the GraphSAGE model generally solves faster while having a similar model accuracy, we chose to use the GraphSAGE configuration. In Fig 14 we can see the plots comparing the mean MSE with the solving time and optimality gaps. For the case study we are only interested to see which molecules could come out of the model. We figured that the `1 x 16` configuration find a good trade off between solving time and model accuracy. The solving time for the `0` configuration is better for GraphSAGE however, its model accuracy is far worse.

B.2 Extra Runs

While experimenting, we figured that the parameters found from training the GNN heavily influenced the solving time of the MILP formulation. To test this hypothesis we ran one of the configurations 5 times and plotted their solving times. We chose the configuration with 2 hidden layers and 16 nodes. This is because this was one of the configurations where the GCN model solved to optimality faster than the GraphSAGE model, which is unexpected. The five trained GNNs per model where randomly trained GNNs. The results can be seen below:

As can be seen in Fig. 15 results are very different depending on which trained GNN is used, for both the GCN model and the GraphSAGE model. We can conclude that the parameters extracted from the trained GNN influence the solving time significantly.

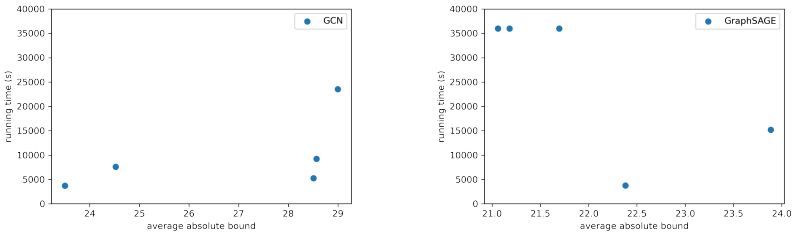
To check whether there was a correlation between the bounds and the solving time we made a scatter plot



(a) GCN (b) GraphSAGE

Figure 15: Bar graphs of the solving time of 5 randomly chosen trained models for both the GCN and GraphSAGE models, with node depth 16 and 2 hidden layers.

comparing the bounds and the solving time. The bounds are the mean absolute bound of the GNN neurons. The results can be found in Fig. 16.



(a) GCN (b) GraphSAGE

Figure 16: Scatter plots comparing the solving time and absolute average bound of 5 randomly chosen trained models for both the GCN and GraphSAGE models, with node depth 16 and 2 hidden layers.

We find no correlation between the bounds and solving time for both models. Further investigation is needed to see what influences the solving time.