Mixed-integer optimisation of graph neural networks for computer-aided molecular design

McDonald, Tom; Tsay, Calvin; Schweidtmann, Artur M.; Yorke-Smith, Neil

DOI
10.1016/j.compchemeng.2024.108660

Publication date
2024

Document Version
Final published version

Published in
Computers and Chemical Engineering

Citation (APA)
McDonald, T., Tsay, C., Schweidtmann, A. M., & Yorke-Smith, N. (2024). Mixed-integer optimisation of graph neural networks for computer-aided molecular design. *Computers and Chemical Engineering*, *185*, Article 108660. https://doi.org/10.1016/j.compchemeng.2024.108660

Computers and Chemical Engineering 185 (2024) 108660

Contents lists available at ScienceDirect

Computers and Chemical Engineering

journal homepage: www.elsevier.com/locate/cace

Mixed-integer optimisation of graph neural networks for computer-aided molecular design

Tom McDonald [a,d], Calvin Tsay [b], Artur M. Schweidtmann [c], Neil Yorke-Smith [d,*]

[a] Delft Institute of Applied Mathematics, Delft University of Technology, Delft, 2600GA, Zuid-Holland, The Netherlands
[b] Department of Computing, Imperial College London, London, SW7 2AZ, England, United Kingdom
[c] Department of Chemical Engineering, Delft University of Technology, Delft, 2600GA, Zuid-Holland, The Netherlands
[d] STAR Lab, Delft University of Technology, Delft, 2600GA, Zuid-Holland, The Netherlands

ARTICLE INFO

Keywords:
Graph neural networks
Mixed integer programming
Optimal boiling point
GraphSAGE
Molecular design

ABSTRACT

ReLU neural networks have been modelled as constraints in mixed integer linear programming (MILP), enabling surrogate-based optimisation in various domains and efficient solution of machine learning certification problems. However, previous works are mostly limited to MLPs. Graph neural networks (GNNs) can learn from non-euclidean data structures such as molecular structures efficiently and are thus highly relevant to computer-aided molecular design (CAMD). We propose a bilinear formulation for ReLU Graph Convolutional Neural Networks and a MILP formulation for ReLU GraphSAGE models. These formulations enable solving optimisation problems with trained GNNs embedded to global optimality. We apply our optimisation approach to an illustrative CAMD case study where the formulations of the trained GNNs are used to design molecules with optimal boiling points.

1. Introduction

The modelling and designing of molecules have long been an interest to researchers. 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. Whereas these methods mostly relied on human expertise and experimentation, Computer Aided Molecular Design (CAMD) has become the de facto state of the art (Achenie et al., 2002). CAMD methods for instance pre-screen a large number of molecules, such that the most promising candidates can be investigated for further testing, saving time and resources.

An early and still established method used for CAMD is the Quantitative Structure Property Relationship (QSPR). With QSPR, chemical descriptors are designed and then used to predict chemical properties (de Lima Ribeiro and Ferreira, 2003; Katritzky et al., 1995). Note that QSPR is used for the formulation of CAMD problems: it is a property prediction model, rather than a solution method. Many examples exist in which QSPR regressions include constitutional (Katritzky et al., 1995; Ha et al., 2005), topological (Begam and Kumar, 2016; de Lima Ribeiro and Ferreira, 2003), electrostatic (Wessel and Jurs, 1995; Egolf et al., 1994), geometrical (Ivanciuc et al., 2002) and quantum-chemical (de Lima Ribeiro and Ferreira, 2003; Hilal et al., 2003) descriptors or combinations of these descriptors. A drawback of QSPR methods is that they are heavily dependent on the knowledge of researchers to select which chemical descriptors are important. In addition, these methods design a molecule in the descriptor space and thus also give a solution in the descriptor space; mapping this vector back to (existing) molecules is highly problematic. Group contribution (Gani et al., 1991; Zhang et al., 2015) presents an alternative family of modelling methodologies (note again, modelling, not solving), where compounds are represented as a collection of functional groups. We note that group contribution methods are sometimes classified as a special case of QSPR regression (Alshehri et al., 2020; Gani, 2019).

In many applications, it is desired to not only predict molecular properties but also perform an inverse design where molecules with desired or optimal properties are identified. This optimisation could also be in the context of additional constraints or even joint process/product optimisation. In a two-step approach, the QSPR regression model is first optimised (e.g., using deterministic or stochastic solvers). In the second step, the (optimal) descriptor vector is mapped back to a molecular structure (e.g., based on enumeration to minimise a distance in the encoding space or other decoding strategies). However, the decoding is often challenging. On the other hand, physically motivated descriptors such as group contribution can enable optimisation of the molecule and

* Corresponding author.
E-mail addresses: thjn.mcdonald@gmail.com (T. McDonald), c.tsay@imperial.ac.uk (C. Tsay), A.Schweidtmann@tudelft.nl (A.M. Schweidtmann), n.yorke-smith@tudelft.nl (N. Yorke-Smith).

https://doi.org/10.1016/j.compchemeng.2024.108660
Received 30 November 2023; Received in revised form 9 March 2024; Accepted 11 March 2024
Available online 22 March 2024

properties together (Folić et al., 2007; Zhang et al., 2015), removing the need for the decoding step.

The advent of machine learning (ML) models and the increased availability of large data sets, resulted in an increased interest in using ML for property prediction tasks (Alshehri et al., 2020). There have been various applications of machine learning in CAMD. Most instances use multi-layer perceptrons (MLPs) in the QSPR methods, where linear or polynomial regression methods are replaced by MLPs to perform regression (Austin et al., 2016). More recently, graph neural networks (GNNs) have been developed for non-euclidean input data types such as molecular graphs. GNNs are neural networks that learn using a graph as input. 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 (Wu et al., 2020). This allows GNNs to take spatial information into consideration when learning non-euclidean data.

The attraction of using GNN in CAMD is that molecules can naturally 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. Moreover, GNNs preserve invariance of the graph structure, e.g., rotating a molecule does not affect the prediction. There have been multiple studies where GNNs have been used to predict properties of molecules (see Wieder et al. (2020) 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. (2022b) have done exactly that, using Bayesian optimisation and a genetic algorithm to optimise the trained GNNs. However, these methods are not deterministic optimisation methods. This means that the 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.

GNNs in chemistry can be categorised into three subgroups (Wieder et al., 2020): (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 to this paper. All of the previously mentioned graph structures have been applied to learning chemical properties. This includes basic Rec-GNNs (Lusci et al., 2013; Scarselli et al., 2008) and gated variants (Mansimov et al., 2019; Withnall et al., 2020; Altae-Tran et al., 2017; Bouritsas et al., 2022). Several Conv-GNNs have also found applications in chemistry, such as spectral Conv-GNNs (Liao et al., 2019; Henaff et al., 2015) and basic (Duvenaud et al., 2015; Errica et al., 2019), attention (Hu et al., 2019) and general (Gilmer et al., 2017) spatial conv-GNNs. For a comprehensive overview of molecular property prediction with graph neural networks, see Wieder et al. (2020).

Recently, various MILP formulations have been introduced for Rectified Linear Unit (ReLU) MLPs (Anderson et al., 2020; Fischetti and Jo, 2018; Huchette et al., 2023; Tsay et al., 2021). ReLU MLPs are MLPs where each activation function is a piece-wise linear function called the ReLU function. Due to its piece-wise linear nature, the activation function can be expressed with linear programming constraints using big-M constraints. The other functions in a MLP are affine and thus the whole network can be linearised. Besides MLP formulations, NNs have also been solved using deterministic global solvers in a reduced space formulation (Schweidtmann and Mitsos, 2019). The class of MILP problems can be solved to global optimality using commercial solvers. This young research area has been applied to a wide variety of topics like MLP verification (Fischetti and Jo, 2018; Tjeng et al., 2017; Bunel et al., 2018; Dutta et al., 2018), compression of MLPs (Kumar et al., 2019; Serra et al., 2020) and using MLPs as surrogate models in linear programming problems (Grimstad and Andersson, 2019; Di Martino et al., 2022; Kody et al., 2022; Yang et al., 2021). We refer the interested reader to Huchette et al. (2023) for an overview of methodologies and applications.

The current work, first reported in the master thesis of McDonald (2022), is to our knowledge the first MILP formulation of a trained GNN presented in the literature. The importance of such a model is 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. More recent work by Zhang et al. (2023) develops symmetry-breaking constraints that can reduce the search space for MILP or other optimisation strategies. Furthermore, there are broader applications, namely the use of MILP formulations of GNNs for similar applications as MLPs (Huchette et al., 2023), e.g., verification of GNNs, lossless compression of GNNs, and using GNNs as surrogate models in optimisation problems. The latter may be of particular interest for applications such as integrated molecule and process design (Bardow et al., 2010).

In particular, this current paper considers two GNN architectures. The first is the Graph Convolutional Neural Network by Kipf and Welling (2017). This neural network is one of the earliest GNN and is used often in GNN applications. The second is the GraphSAGE network by Hamilton et al. (2017), which learns properties of large graph data by sampling the neighbourhood of nodes instead of using information of all neighbouring nodes. In line with much of the literature, the formulations proposed are for fixed N, i.e., number of nodes in the graph or number of atoms in the molecule.

Contributions. Summarised, this paper adds to the state-of-the-art in the literature as follows:

- We propose a mixed integer quadratically constrained programming formulation of the frequently used Graph Convolutional Network model by Kipf and Welling (2017).
- We propose a mixed integer linear programming formulation of the GraphSAGE model by Hamilton et al. (2017).
- We demonstrate the computational performance of our approach on a case study of optimising the boiling points of molecules modelled with the GraphSAGE and GCN models.

Organisation. Following this introduction, Section 2 provides technical background, leading to our main contribution of the MI(N)LP formulations of GNNs in Section 3. Section 4 reports empirical results on a case study. Section 5 discusses the models and results, and Section 6 concludes.

2. Background

This section introduces the terminology for neural networks needed in the remainder of the paper. We assume the reader has familiarity with mixed integer (linear) programming, referring to Wolsey (2020) for an introduction.

2.1. Multilayer perceptrons

A feedfoward multilayer perceptron (MLP) consists of consecutive layers of neurons connected through a directed acyclic graph. A neuron in a particular layer receives 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 reaches a particular threshold. The result of this system is a neural network that 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 $k \in \{1, \dots, K\}$, including the input layer $k = 1$, the hidden layers $k = \{2, \dots, K-1\}$ and the output layer $k = K$. Each layer contains 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 $b^k \in \mathbb{R}^{n_k}$ (Goodfellow et al., 2016).

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 non-linear activation functions. 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 $k = \{2, \dots, K-1\}$ we have

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

where $\sigma(\cdot)$ is the activation function. Normally, in the last layer K the activation function is absent. Completely composed, the neural network $f(x) : \mathbb{R}^{n_1} \mapsto \mathbb{R}^{n_K}$ is defined by (Goodfellow et al., 2016):

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

The activation function, indicated by σ in Eq. (1), is a non-linear function, which allows the neural network to find a non-linear relationship between input and output data. Commonly used activation functions include the sigmoid, tanh, and ReLU functions. The latter will be the main focus for this contribution. It is defined as $\sigma(z) = \max\{0, z\}$.

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 weights and biases to minimise a loss function, e.g., mean squared error (MSE), for the predictions and target values (Goodfellow et al., 2016).

2.2. MILP formulations of multilayer perceptrons

Exact MILP formulations of NNs with ReLU activation functions have been proposed. These exact formulations emulate the ReLU operator using binary activation variables and big-M formulations. We refer to Huchette et al. (2023) for a survey of methods and applications and defer some details to Appendix A.

Consider, for each hidden layer $k \in \{1, \dots, K-1\}$, the following MLP layer:

$$x^k = \sigma(W^k x^{k-1} + b^k) \tag{3}$$

where $\sigma(\cdot) = \max\{0, \cdot\}$ 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. This paper considers the linearisation of (3) by Fischetti and Jo (2018). The output of the affine equations are decoupled in a positive part $x \geq 0$ and negative part $s \geq 0$, and a binary activation variable z and big-M activation constraints are introduced. 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 is introduced:

$$x_j^k \leq u_j^k z_j^k \tag{4a}$$

$$s_j^k \leq -l_j^k (1 - z_j^k) \tag{4b}$$

$$z_j^k \in \{0, 1\}. \tag{4c}$$

The big-M constraints are applied to every node in the network.

2.3. Graph neural networks

We now turn from MLPs and 'regular' neural networks to GNNs.

2.3.1. General graph neural network architecture

When using GNNs for property prediction, each 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. For our purposes of CAMD, node features will suffice. 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 $((X, A), y)$.

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 sums 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 $\forall s \in \mathcal{N}(i)$. This works as follows: in the first convolutional layer, for every node i, all neighbourhood information is aggregated. In the next layer this is repeated; 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 subsections we will discuss the graph aggregation functions of two GNNs, for they define the architectures of the GNNs we consider (see Fig. 1).

2.3.2. Graph Convolutional Neural network (GCN)

We focus on spatial Conv-GNN methods, which are conceptually similar to non-graph based convolutional neural nets (CNN), as "spatial-based graph convolutions convolve the central node's representation with its neighbours' representations to derive the updated representation for the central node" (Wu et al., 2020). Micheli (2009) 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 (Niepert et al., 2016; Gao et al., 2018; Hamilton et al., 2017). 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 (Velickovic et al., 2018; Zhang et al., 2018). Finally, there are more general frameworks which try to unify multiple models in a single formulation as an abstraction over multiple GNNs (Monti et al., 2017; Gilmer et al., 2017; Battaglia et al., 2018).

The first GNN we consider is the Graph Convolutional Neural Network (GCN) (Kipf and Welling, 2016, 2017). It is one of the earlier models 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 (Defferrard et al., 2016) architecture, which is a spectral based method. However, this first order approximation is basically a spatial based method.

Kipf and Welling (2017) introduce the kth convolutional layer in the GCN can be expressed as follows:

$$H^{(k+1)} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^k W^k) \tag{5}$$

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 outputs from the activation functions in the kth 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:

$$H_{ij}^{(k+1)} = \sigma\left(\left(W_j^k\right)^T \sum_{l \in N^+(i)} \frac{1}{\sqrt{d^+(i)}\sqrt{d^+(l)}} \left(H_l^k\right) \right) \tag{6}$$

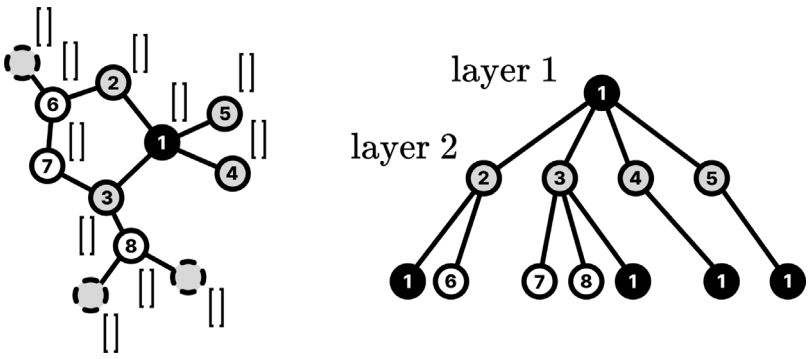


Fig. 1. A graph (left) with a feature vector on every node. Neighbourhoods of a node with multiple convolutional layers in a spatial GNN (right).

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.

2.3.3. GraphSAGE network

The GraphSAGE network is another spatial convolutional neural network, 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 paper 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 do not 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:

$$H_i^{(k+1)} = \sigma\left(H_i^k \cdot W_1^k + \sum_{l \in \mathcal{N}(i)} H_l^k \cdot W_2^k \right) \tag{7}$$

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 subsection.

3. Methods

This section provides the main contribution of the paper, by presenting the novel formulations of graph neural networks as MI(N)LPs, starting from the multilayer perceptron MILP formulation of Section 2.2. Section 3.1 describes a formulation of the Graph Convolutional Network, which is linear (MILP) for a fixed graph structure and bi-linear (MINLP, specifically, MIQCP) for a variable graph structure. Then, Section 3.2 describes a MILP formulation of the GraphSAGE architecture. Section 3.3 shows how to add domain-specific background knowledge (inductive bias). Finally, Section 3.4 applies bound tightening techniques to the formulations.

3.1. MI(N)LP formulation for GCN models

We wish to train a graph neural network that finds the function $f : \{0,1\}^{|N|\times|N|} \times \mathbb{R}^{|N|\times|F|} \mapsto \mathbb{R}$. Note we make the common assumption of fixed N, i.e., number of nodes in the graph or number of atoms in the molecule. The function f 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:

$$f(A, X) = \mathsf{MLP}(\mathsf{POOL}(\mathsf{GNN}(A, X))) \tag{8}$$

where POOL is the pooling layer. This subsection states the linear MILP formulations for graph neural networks in case the graph structure is predetermined. Note that the predetermined graph structure is not directly relevant for molecular design but rather for other applications and domains where the topology is fixed and other parameters may be optimised (e.g., optimisation of chemical process operation where the flowsheet topology is represented as graph Stops et al., 2023). Further, this MILP formulation is a development step towards the MIQCP formulation the subsequent section.

3.1.1. Predetermined graph structure

In the following subsection the Graph Convolutional Network layers as described in Section 2.3 are formulated as an MILP. We first consider the GCN, which has the layer-wise propagation rule defined by Eq. (5). To linearise Eq. (5) we employ the big-M formulation stated in Eqs. (A.5). As described in Section 2.2, 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 kth layer. For the GCN structure we have the same but for every node $i \in \{1, \dots, N\}$. 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 Eqs. (A.5b) and (A.5c) need to be altered to represent the linear part of the GCN layer as described between the brackets in Eq. (5). 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:

$$\bar{A}_{il} = \begin{cases} 0, & \text{if } \tilde{A}_{il} = 0 \\ \frac{1}{\sqrt{d^+(i)}\sqrt{d^+(l)}}, & \text{if } \tilde{A}_{il} = 1 \end{cases} \tag{9}$$

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 (2016, 2017) note the following, with N nodes in our graph

$$Y^0 = X = \begin{bmatrix} X_1 \\ \vdots \\ X_N \end{bmatrix} \tag{10}$$

where X_i is a row vector containing the features of node i. H^k_{ij} is considered, which is the jth 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 for clarity):

$$H^k_{ij} = \left(\tilde{A}H^{(k-1)}W^k\right)_{ij} \tag{11a}$$

$$= \tilde{A}_i \begin{bmatrix} H^{(k-1)}_1 W^k_j \\ \dots \\ H^{(k-1)}_N W^k_j \end{bmatrix} \tag{11b}$$

$$= \sum_{l \in N^+(i)} \frac{1}{\sqrt{d^+(i)}\sqrt{d^+(l)}} \left(W^k_j\right)^T \left(H^{(k-1)}_l\right)^T \tag{11c}$$

It is commonplace to write vectors in column notation for linear programming, so we will deviate from Kipf and Welling (2017), replacing $\left(H^{(k-1)}_l\right)^T$ by $\left(H^{(k-1)}_l\right)$. The MILP formulation becomes:

$$\sum_{l|\tilde{A}_{il}=1} \frac{1}{\sqrt{d^+_i d^+_l}} {W^k_j}^T H^{(k-1)}_l = H^k_{ij} - S^k_{ij}$$

$$\forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{12a}$$

$$H^k_{ij} \le U^k_{ij} Z^k_{ij} \qquad \forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{12b}$$

$$S^k_{ij} \le -L^k_{ij}(1 - Z^k_{ij}) \qquad \forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{12c}$$

$$0 \le H^k_{ij}, S^k_{ij} \in \mathbb{R} \qquad \forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{12d}$$

$$Z^k_{ij} \in \{0,1\} \qquad \forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{12e}$$

$$A, H^0 \in \Omega \tag{12f}$$

where i indicates node i, j feature j, and k the corresponding GCN layer. d^+_i represents the degree +1 of node i. The summation over $\tilde{A}_{il} = 1$ in (12a) becomes a conditional sum if the graph structure is not fixed (and the entries of $\tilde{A}$ become decision variables); this is discussed in the next subsection.

In the above, $H^0 \in \Omega$ indicates a restriction of the input space (see Section 3.3). For a molecule we can view these as constraints which provide domain knowledge: the knowledge that only physically possible molecules are considered in the input space. In this regard they lead simply to a form of physics-informed neural networks (Razakh et al., 2021). One example could be that node i, has a feature $x_{ij} = 1$ indicating that it is a carbon atom. In that case $\sum_j A_{ij} < 5$, meaning the number of bonds should be exactly four (counting a double bond as 2 and a triple bond as 3.

Combining all constraints of (12) with (4) almost finalises our formulation of function (8) 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 to maintain a linear model:

$$x^0_j = \sum_{i=1}^{N} H^K_{ij} \quad \forall j \in \{1,\dots,n_K\} \tag{13}$$

Naturally, the number 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.

3.1.2. Variable graph structure

The formulation of the previous subsection 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 dependent on the former non-linear term. We describe a bi-linear formulation for these terms, which MIQP/MIQCP solvers such as Gurobi can accommodate.

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

$$b^k_{il} = \begin{cases} 0 & \text{if } \tilde{A}_{il} = 0 \\ H^k_l & \text{if } \tilde{A}_{il} = 1 \end{cases} \tag{14}$$

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

The logic as described in (14) can be implemented in the same way as was done in Eq. (A.3) by using big-M constraints, because the entries of $\tilde{A}$ are binary. Replacing Eq. (12a) 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:

$$\sum_{l=1}^{N} \frac{1}{\sqrt{d^+_i d^+_l}} {W^k_j}^T b^{(k-1)}_{il} = H^k_{ij} - S^k_{ij} \tag{15a}$$

$$H^{(k-1)}_l - M(1 - \tilde{A}_{il}) \le b^{(k-1)}_{il} \le H^{(k-1)}_l + M(1 - \tilde{A}_{il}) \tag{15b}$$

$$- M(\tilde{A}_{il}) \le b^{(k-1)}_{il} \le M(\tilde{A}_{il}) \tag{15c}$$

In case node i is not connected to node l, then $\tilde{A}_{il} = 0$. In that case constraint (15c) forces $b^{(k-1)}_{il} = 0$. In case both nodes are connected, $\tilde{A}_{il} = 1$ and $b^{(k-1)}_{il}$ is constrained by (15b) such that it is equal to $H^{(k-1)}_l$.

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^k_{il}, which makes the entire constraint non-linear. While an auxiliary variable formulation could be employed (Vielma, 2015a), this would result in many 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 bounded above by the maximum degree of the graph d_{max}, adding 1 for the self loops. In case of molecules, the maximum covalence and thus the maximum degree, 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 linearise the fractional term in Eq. (15) by the following set of equations

$$\sum_{l=1}^{N} s_{il} {W^k_j}^T b^{(k-1)}_{il} = H^k_{ij} - S^k_{ij}$$

$$\forall k \in \{1,\dots,K\}, \forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{16a}$$

$$d^+_i = \sum_j \tilde{A}_{ij} \qquad \forall i \in \{1,\dots,N\} \tag{16b}$$

$$p_{il} = d^+_i(d_{max}+1) + d^+_l \qquad \forall i,l \in \{1,\dots,N\} \tag{16c}$$

$$0 = p_{il} - 1c^{il}_1 - 2c^{il}_2 - \cdots \tag{16d}$$

$$- (d_{max}+1)^2 c^{il}_{(d_{max}+1)^2} \qquad \forall i,l \in \{1,\dots,N\} \tag{16e}$$

$$1 = c^{il}_1 + \cdots + c^{il}_{(d_{max}+1)^2} \qquad \forall i,l \in \{1,\dots,N\} \tag{16f}$$

$$c^{il} \in \{0,1\}^{(d_{max}+1)^2} \qquad \forall i,l \in \{1,\dots,N\} \tag{16g}$$

$$s_{il} = c^{il}_1 g_1 + \cdots + c^{il}_{(d_{max}+1)^2} g_{(d_{max}+1)^2} \qquad \forall i,l \in \{1,\dots,N\} \tag{16h}$$

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. The resulting model is bi-linear as it involves multiplication of decision variables s_{il} and b_{il}.

3.1.3. Full MIQCP formulation of the GCN GNN

Section 3.1.2 presents a reformulation for the non-linear terms of Eq. (12a). Specifically, the conditional sum is reformulated as Eq. (15) and the normalisation term as Eq. (16), resulting in an overall bi-linear formulation. For GCN models, we can combine the binary variables introduced by Eqs. (15) and (16) Notice how for every layer k, Eqs. (15b) and (15c) 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 $i, l \in \{1, \dots, N\}$:

$$\hat{s}_{il} = \begin{cases} 0 & \text{if } \tilde{A}_{il} = 0 \\ s_{il} & \text{if } \tilde{A}_{il} = 1 \end{cases} \tag{17}$$

This enforces that the feature vector of a neighbouring node of i is only included if $\tilde{A}_{il} = 1$, which is the same as $\sum_{l|\tilde{A}_{il}=1}$. The resulting bi-linear mixed-integer quadratically constrained programming (MIQCP) formulation becomes:

$$\sum_{l=1}^{N} \hat{s}_{il} {W_j^k}^T H_l^{(k-1)} = H_{ij}^k - S_{ij}^k$$
$$\forall k \in \{1, \dots, K\}, \forall i \in \{1, \dots, N\}, \forall j \in \{1, \dots, n_k\} \tag{18a}$$

$$d_i^+ = \sum_{j=1}^{n_k} \tilde{A}_{ij} \qquad \forall i \in \{1, \dots, N\} \tag{18b}$$

$$p_{il} = d_i^+ (d_{max} + 1) + d_l^+ \qquad \forall i, l \in \{1, \dots, N\} \tag{18c}$$

$$0 = p_{il} - 1c_1^{il} - 2c_2^{il} - \cdots \tag{18d}$$

$$- (d_{max} + 1)^2 c^{il}_{(d_{max}+1)^2} \qquad \forall i, l \in \{1, \dots, N\} \tag{18e}$$

$$1 = c_1^{il} + \cdots + c^{il}_{(d_{max}+1)^2} \qquad \forall i, l \in \{1, \dots, N\} \tag{18f}$$

$$c^{il} \in \{0, 1\}^{(d_{max}+1)^2} \qquad \forall i, l \in \{1, \dots, N\} \tag{18g}$$

$$s_{il} = c_1^{il} g_1 + \cdots + c^{il}_{(d_{max}+1)^2} g_{(d_{max}+1)^2} \qquad \forall i, l \in \{1, \dots, N\} \tag{18h}$$

$$s_{il} - M(1 - A_{il}) \le \hat{s}_{il} \le M(1 - A_{il}) + s_{il} \qquad \forall i, l \in \{1, \dots, N\} \tag{18i}$$

$$- M A_{il} \le \hat{s}_{il} \le M A_{il} \qquad \forall i, l \in \{1, \dots, N\} \tag{18j}$$

In this formulation, constraints (18b)–(18h) describe the linearisation of the normalisation term as described in Section 3.1.2 and constraints (18i) and (18j) incorporate the conditional-sum logic from Eq. (17). Note that we only have to find $\hat{s}_{il}$ once for every layer, since the structure of the molecule is constant per layer.

Once again we incorporate the logical connectives with big-M constraints. One can in principle use something like interval arithmetic to get reasonable M values. Later, one can tighten them further through optimisation-based bound tightening.

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. (18i) and (18j) 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.

3.2. GraphSAGE

So far we have successfully formulated an exact MIQCP (bi-linear) representation of a GCN. This section describes an MILP formulation for a more recent and popular GNN architecture, the *GraphSAGE* model by Hamilton et al. (2017).

In this paper the activation function and affine layer are described by the following equation:

$$f^{(t)}(v) = \sigma\left(f^{(t-1)}(v) \cdot W_1^{(t)} + \sum_{w \in N(v)} f^{(t-1)}(w) \cdot W_2^{(t)} \right) \tag{19}$$

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

As can be seen from Eq. (19), 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 of the model is linear.

For consistency, we rewrite Eq. (19) in a notation similar to the presentation in Section 3.1. We find the following for node $i \in \{1, \dots, N\}$, feature $j \in \{1, \dots, n_k\}$ and layer $k \in \{1, \dots, K\}$:

$$H_{ij}^k = \sigma\left((\hat{W}_j^k)^T H_i^{(k-1)} + (\bar{W}_j^k)^T \sum_{l|A_{il}=1} H_l^{(k-1)} \right) \tag{20}$$

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:

$$b_{il}^k = \begin{cases} 0 & \text{if } A_{il} = 0 \\ H_l^k & \text{if } A_{il} = 1 \end{cases} \tag{21}$$

The full MILP formulation including the pooling layer becomes:

$$(\hat{W}_j^k)^T H_i^{(k-1)} + (\bar{W}_j^k)^T \sum_l b_{il}^{(k-1)} = H_{ij}^k - S_{ij}^k$$
$$\forall k \in \{1, \dots, K\}, \forall i \in \{1, \dots, N\}, \forall j \in \{1, \dots, n_k\} \tag{22a}$$

$$H_{ij}^k \le U_{ij}^k Z_{ij}^k \qquad \forall k \in \{1, \dots, K\}, \forall i \in \{1, \dots, N\}, \forall j \in \{1, \dots, n_k\} \tag{22b}$$

$$S_{ij}^k \le -L_{ij}^k (1 - Z_{ij}^k) \qquad \forall k \in \{1, \dots, K\}, \forall i \in \{1, \dots, N\}, \forall j \in \{1, \dots, n_k\} \tag{22c}$$

$$H_l^k - M(1 - A_{il}) \le b_{il}^k \qquad \forall k \in \{0, \dots, K-1\}, \forall i, l \in \{1, \dots, N\} \tag{22d}$$

$$b_{il}^k \le H_l^k + M(1 - A_{il}) \qquad \forall k \in \{0, \dots, K-1\}, \forall i, l \in \{1, \dots, N\} \tag{22e}$$

$$- M(A_{il}) \le b_{il}^k \le M(A_{il}) \qquad \forall k \in \{0, \dots, K-1\}, \forall i, l \in \{1, \dots, N\} \tag{22f}$$

$$H_i^0 = x_i \qquad \forall i \in \{1, \dots, N\} \tag{22g}$$

$$H_j^{*K} = \sum_i H_{ij}^K \qquad \forall j \in \{1, \dots, n_K\} \tag{22h}$$

$$0 \le H_{ij}^k, S_{ij}^k \in \mathbb{R} \qquad \forall k \in \{1, \dots, K\}, \forall i \in \{1, \dots, N\}, \forall j \in \{1, \dots, n_k\} \tag{22i}$$

$$Z_{ij}^k \in \{0, 1\} \qquad \forall k \in \{1, \dots, K\}, \forall i \in \{1, \dots, N\}, \forall j \in \{1, \dots, n_k\} \tag{22j}$$

$$x, A \in \Omega \tag{22k}$$

Intuitively, Eqs. (22a)–(22c) reformulate the ReLU function in Eq. (20), Eqs. (22d)–(22f) are the big-M constraints to enforce the logic in Eq. (21). The values of these big-M constraints are the same as the upper bounds U_{ij}^k (as explained in Section 3.4.3). Eq. (22g) defines the input feature vector x_i of node i and Eq. (22h) is the sum pooling layer in layer K. Finally, Eq. (22k) represent the input constraints as described in Section 3.3.

Note that a drawback of this method compared to the MIQCP formulation is that constraints (22d), (22e) and (22f) are calculated for every layer k. This increases the number of constraints significantly (by $\mathcal{O}(n^2 n_k)$ constraints per layer k). For the GCN network this is only

Table 1
Background knowledge about molecule properties.

X_{if}	Type	Descriptor
1	Atom	C
2	Atom	O
3	Atom	F
4	Atom	Cl
5	Neighbours	0
6	Neighbours	1
7	Neighbours	2
8	Neighbours	3
9	Neighbours	4
10	Hydrogen	0
11	Hydrogen	1
12	Hydrogen	2
13	Hydrogen	3
14	Hydrogen	4

$\mathcal{O}(n^2)$. For networks where there are a lot of nodes per hidden layer, the GraphSAGE network will have significantly more constraints than the GCN network.

3.3. Constraining the input space for molecular design

With the purpose of CAMD in mind, we next describe the input space constraints, $A, x \in \Omega$. These constraints limit the search space to include structures which try to emulate physically-feasible molecular structures.

3.3.1. Basic MILP formulation of molecules

QSPR methods used for property prediction in previous works are mostly based on group contribution methods (Zhang et al., 2015). As a result MILP formulations of molecules used in CAMD are also often based on group contribution methods. 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 therefore introduce an MILP formulation for molecules based on topological structure similar to the input of GNNs. This is similar in spirit to topological indexing methods for QSPR (Austin et al., 2016), which are in turn based on chemical graph theory.

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 we consider comprises 14 features, which represent the knowledge summarised in Table 1. The first 4 entries of the vector x_{if} indicate the atom type of atom i. Positions 5 to 9 indicate the number of neighbours atom i has. Finally, the last entries show how many hydrogen atoms are connected to atom i. For example, a solution can be found where node i is an oxygen atom, with one neighbour and one hydrogen atom attached. In this case, for atom i, $x_{if} = 1$ for $f \in \{2, 6, 11\}$ and 0 for the other values of f.

With the adjacency matrix and the feature vectors for all the nodes, we introduce constraints to avoid trivially infeasible molecules. These constraints are summarised here and detailed in Appendix C:

(a) Molecules should be connected and of at least length 2.
(b) Nodes are active if and only if they are connected to others.
(c) To avoid redundancies, no gaps should exist between activated atoms (a molecule of length 3 should be $A_{11} = A_{22} = A_{33} = 1$ and not $A_{11} = A_{22} = A_{55} = 1$).
(d) Each node should only have one atom type.
(e) The covalence of each atom must equal the number of active neighbours.

The 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 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. Therefore we next describe how to extend this model.

3.3.2. Additional properties

The formulation in the previous subsection is seen as a basis which can be extended or modified for a particular CAMD setting. To simplify the model, we limit the search space to molecules with no loops. We can achieve this with the following constraint:

$$\sum_{i}^{n-1} \sum_{j|j>i}^{n} A_{ij} = n - 1 \tag{23}$$

The constraint guarantees that the number of edges (LHS) is equal to the number of nodes minus one. When added to the set of constraints (C.1), 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 (C.1l). This fact, combined with basic graph properties and the result of constraint (23), guarantees the graph to be acyclic.

Next, we consider constraints to have the search space include double bonded molecules. These are molecules for which there are two bonds between two connected atoms in the molecule. To find solutions with 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, we leave such network architectures for subsequent work.

We introduce a binary variable db_{il} that tracks whether a double bond is present between nodes i and l.

$$3 \cdot db_{il} \le x_{i,15} + x_{l,15} + A_{il} \qquad \forall i, l \in \{1, \dots, n\} \tag{24a}$$

$$2 \cdot x_{i,1} + 1 \cdot x_{i,2} \ge \sum_{l}^{n} db_{il} \qquad \forall i \in \{1, \dots, n\} \tag{24b}$$

$$4 \cdot x_{i,1} + 2 \cdot x_{i,2} + 1 \cdot x_{i,3} + 1 \cdot x_{i,4} = \tag{24c}$$

$$\sum_{s=0}^{4} s \cdot x_{i,(5+s)} + \sum_{s=0}^{4} s \cdot x_{i,(10+s)} + \sum_{l}^{n} db_{il} \qquad \forall i \in \{1, \dots, n\} \tag{24d}$$

$$x_{i,15} \le \sum_{l}^{n} db_{il} \qquad \forall i \in \{1, \dots, n\} \tag{24e}$$

$$db_{il} = db_{l,i} \qquad \forall i, l \in \{1, \dots, n\} \tag{24f}$$

$$db_{i,i} = 0 \qquad \forall i \in \{1, \dots, n\} \tag{24g}$$

Constraint (24a) enforces that double bonds are only possible if nodes i and l are connected and $x_{i,15} = x_{l,15} = 1$. Constraint (24b) limits the number of double bonds based on the covalence of the atom. For instance, $x_{i,1}$ indicates a carbon atom, meaning that there can be a maximum of 2 double bonds. Constraint (24c) limits the total number of bonds (including double bonds). Constraint (24e) forces $x_{i,15}$ to be zero if there are no double bonds connected to node i. 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 (24b) would have $1 * x_{i,1}$ on the left-hand side because generally, only a carbon atom can have a triple bond. Finally, in constraint (24c), $\sum_{l}^{n} tb_{il}$ 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.

$$3 \cdot db_{il} \le x_{i,15} + x_{l,15} + A_{il} \qquad \forall i, l \in \{1, \dots, n\} \tag{25a}$$

$$3 \cdot tb_{il} \leq x_{i,16} + x_{l,16} + A_{il} \qquad \forall i,l \in \{1,\dots,n\} \qquad (25b)$$

$$2 \cdot x_{i,1} + 1 \cdot x_{i,2} \geq \sum_{l}^{n} db_{il} \qquad \forall i \in \{1,\dots,n\} \qquad (25c)$$

$$1 \cdot x_{i,1} \geq \sum_{l}^{n} tb_{il} \qquad \forall i \in \{1,\dots,n\} \qquad (25d)$$

$$4 \cdot x_{i,1} + 2 \cdot x_{i,2} + 1 \cdot x_{i,3} + 1 \cdot x_{i,4} = \sum_{s=0}^{4} s \cdot x_{i,(5+s)} + \sum_{s=0}^{4} s \cdot x_{i,(10+s)} + \sum_{l}^{n} db_{il} + \sum_{l}^{n} 2 \cdot tb_{il} \qquad \forall i \in \{1,\dots,n\} \qquad (25e)$$

$$x_{i,15} \leq \sum_{l}^{n} db_{il} \qquad \forall i \in \{1,\dots,n\} \qquad (25f)$$

$$x_{i,16} \leq \sum_{l}^{n} tb_{il} \qquad \forall i \in \{1,\dots,n\} \qquad (25g)$$

$$db_{il} = db_{li} \qquad \forall i,l \in \{1,\dots,n\} \qquad (25h)$$

$$tb_{il} = tb_{li} \qquad \forall i,l \in \{1,\dots,n\} \qquad (25i)$$

$$db_{ii} = tb_{ii} = 0 \qquad \forall i \in \{1,\dots,n\} \qquad (25j)$$

$$db_{li} + tb_{li} \leq 1 \qquad \forall i,l \in \{1,\dots,n\} \qquad (25k)$$

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 does not 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.

3.4. 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.

3.4.1. Big-M coefficients for 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 (Gleixner et al., 2017). For the formulation of MLPs in Eq. (4), recall we require bounds such that $l \leq w^T y + b \leq u$. We now denote these as l_j^k and u_j^k for a node j in layer k. 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:

$$u_j^k = \sum_{i=1}^{n_{k-1}} \max\left\{w_{ji}^k \max\left\{0, u_i^{k-1}\right\}, w_{ji}^k \max\left\{0, l_i^{k-1}\right\}\right\} + b_j^k, \qquad (26a)$$

$$l_j^k = \sum_{i=1}^{n_{k-1}} \min\left\{w_{ji}^k \max\left\{0, u_i^{k-1}\right\}, w_{ji}^k \max\left\{0, l_i^{k-1}\right\}\right\} + b_j^k. \qquad (26b)$$

Note that the inner max operators capture the ReLU activation function, i.e., model how ReLU works here, but in the form of linear constraints. 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:

$$\begin{aligned} u_j^k &= \max\left\{t_j^k : t_j^k \in C_j^k\right\} \\ l_j^k &= \min\left\{t_j^k : t_j^k \in C_j^k\right\} \end{aligned} \qquad (27)$$

for the constraint set

$$C_j^k = \left\{t_j^k : t_j^k = w_j^k x^{k-1} + b^k, x^{k-1} \in \left[\max\left\{0, L^{k-1}\right\}, \max\left\{0, U^{k-1}\right\}\right] \subset \mathbb{R}^{n_{k-1}}\right\} \qquad (28)$$

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 must be 1. The same goes for a positive lower bound $l_j^k > 0$: in this case $z_j^k = 0$.

3.4.2. Big-M coefficients for GCNs

The following subsection explains how to find the upper and lower bounds associated with the ReLU constraints for a GCN model. Specifically, we require the upper and lower bounds, U_{ij}^k and L_{ij}^k in Eqs. (12a)–(12c).

It is assumed that the lower and upper bounds 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 the feature vectors must be equal in length. This makes the bound propagation symmetric over all nodes, which in turn 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 the case of maximisation, we have to find a scalar which upper bounds H_{ij}^k for all possible neighbourhood structures of node i. Specifically, we compute d_i^+ and d_l^+ such that 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):

$$\max_{d_i^+, d_l^+} \quad d_i^+ \frac{1}{\sqrt{d_i^+ d_l^+}} {W_j^k}^T H_l^{(k-1)} \qquad (29)$$

The upper bound of ${W_j^k}^T H_l^{(k-1)}$ is determined as in Eq. (26a) with zero bias. If this upper bound is positive, we want to add as much as possible to account for all possible neighbourhood structures, i.e., $\frac{d_i^+}{d_l^+}$ should be maximised. 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 $d_l^+ = 1 + 1$. One of other hand, if the upper bound from Eq. (26a) is negative, $\frac{d_l^+}{d_i^+}$ should be minimised following the same logic. This results in

$$U_{ij}^k = \max\left\{ \sqrt{\tfrac{d_{max}+1}{2}} \sum_{s=1}^{n_{k-1}} \max\left\{w_{js}^k \max\left\{0, U_{sj}^{k-1}\right\}, w_{js}^k \max\left\{0, L_{sj}^{k-1}\right\}\right\}, \sqrt{\tfrac{2}{d_{max}+1}} \sum_{s=1}^{n_{k-1}} \max\left\{w_{js}^k \max\left\{0, U_{sj}^{k-1}\right\}, w_{js}^k \max\left\{0, L_{sj}^{k-1}\right\}\right\} \right\} \qquad (30a)$$

$$L_{ij}^k = \min\left\{ \sqrt{\tfrac{d_{max}+1}{2}} \sum_{s=1}^{n_{k-1}} \min\left\{w_{js}^k \max\left\{0, U_{sj}^{k-1}\right\}, w_{js}^k \max\left\{0, L_{sj}^{k-1}\right\}\right\}, \sqrt{\tfrac{2}{d_{max}+1}} \sum_{s=1}^{n_{k-1}} \min\left\{w_{js}^k \max\left\{0, U_{sj}^{k-1}\right\}, w_{js}^k \max\left\{0, L_{sj}^{k-1}\right\}\right\} \right\} \qquad (30b)$$

In practice, we only need the first term of each bound, as 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 3.4.1).

For a similar formulation as Eqs. (27) and (28), the following set of equations can be considered:

$$U_{ij}^k = \max\left\{t_{ij}^k : t_{ij}^k \in C_{ij}^k\right\} \qquad (31a)$$

$$L_{ij}^k = \min\left\{t_{ij}^k : t_{ij}^k \in C_{ij}^k\right\} \qquad (31b)$$

$$C_{ij}^k = \left\{t_{ij}^k : t_{ij}^k = \sqrt{\frac{d_{max}+1}{2}} w_j^k x^{k-1}, \qquad (31c)$$

$$x^{k-1} \in \left[\max\left\{0, L_i^{k-1}\right\}, \max\left\{0, U_i^{k-1}\right\}\right] \subset \mathbb{R}^{n_{k-1}}\right\} \qquad (31d)$$

3.4.3. GraphSAGE

For GraphSAGE, FBBT is very similar to the FBBT proposed for the GCN. We now require the upper and lower bounds, U_{ij}^k and L_{ij}^k in Eqs. (22a)–(22c).

There are two parts that contribute to the total bound. The first, which is associated with the root node i, is calculated similarly as the MLP FBBT. The second part is calculated in the same way as the GCN, but with the root node i omitted. This results in the following bound propagation equations:

$$U_{ij}^k = \sum_{s=1}^{n_{k-1}} \max\left\{\hat{w}_{js}^k \max\left\{0, U_{sj}^{k-1}\right\}, \hat{w}_{js}^k \max\left\{0, L_{sj}^{k-1}\right\}\right\} + \sqrt{\frac{d_{max}}{2}} \sum_{s=1}^{n_{k-1}} \max\left\{\bar{w}_{js}^k \max\left\{0, U_{sj}^{k-1}\right\}, \bar{w}_{js}^k \max\left\{0, L_{sj}^{k-1}\right\}\right\}, \tag{32a}$$

$$L_{ij}^k = \sum_{s=1}^{n_{k-1}} \min\left\{\hat{w}_{js}^k \max\left\{0, U_{sj}^{k-1}\right\}, \hat{w}_{js}^k \max\left\{0, L_{sj}^{k-1}\right\}\right\} + \sqrt{\frac{d_{max}}{2}} \sum_{s=1}^{n_{k-1}} \min\left\{\bar{w}_{js}^k \max\left\{0, U_{sj}^{k-1}\right\}, \bar{w}_{js}^k \max\left\{0, L_{sj}^{k-1}\right\}\right\}. \tag{32b}$$

Notice that $\sqrt{\frac{d_{max}}{2}}$ replaces $\sqrt{\frac{d_{max}}{2}}$ because the root node is omitted.

For a similar formulation as Eqs. (27) and (28), the following set of equations can be considered:

$$U_{ij}^k = \max\left\{t_{ij}^k : t_{ij}^k \in C_{ij}^k\right\} \tag{33a}$$

$$L_{ij}^k = \min\left\{t_{ij}^k : t_{ij}^k \in C_{ij}^k\right\} \tag{33b}$$

$$C_{ij}^k = \left\{t_{ij}^k : t_{ij}^k = \hat{w}_j^k y^{k-1} + \sqrt{\frac{d_{max}}{2}} \bar{w}_j^k x^{k-1}, \right. \tag{33c}$$

$$\left. x^{k-1}, y^{k-1} \in \left[\max\left\{0, L_i^{k-1}\right\}, \max\left\{0, U_i^{k-1}\right\}\right] \subset \mathbb{R}^{n_{k-1}}\right\} \tag{33d}$$

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 subsection also serve as the values for the big-M constraints in (22d), (22e) and (22f). For instance, consider constraint (22d) 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 M needs to be subtracted such that the left hand side of the equation is lower than 0. To achieve this M needs to be larger than H_l^k, which can be achieved if M is equal to the upper bound of H_l^k.

3.5. Benchmark genetic algorithm

As a benchmark, we implement a straightforward genetic algorithm (GA) to optimise over trained GNNs. This GA does not depend on a latent space architecture, as proposed by Rittig et al. (2022a). It 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. A description is given in Appendix D.

4. Numerical results

Recall that the goal of this work is to formulate graph neural networks such that they can be used as surrogate models in optimisation problems. To validate the MI(N)LP formulations, we turn to the case study of in-silico chemical property prediction, specifically the maximisation of boiling points. Note that the goal here is not to test the quality of the predictions.

4.1. Experimental setup

The workflow of the computer experiments is as follows. First a data set is chosen, which will be used to train a GNN. Thereafter, this trained GNN is represented using the methods described in Section 3. The resulting formulations are optimised using a deterministic solver, namely Gurobi version 9.5.1 (Gurobi Optimization, 2022).

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 virtual machine was used for experiments where solving times were compared. These experiments took longer and required constant CPU availability. The laptop was equipped with a 1.4 GHz Quad-Core Intel Core i5 processor, and 8 GB memory. The virtual machine was equipped with eight 2.5 GHz Intel Xeon Gold 6248 CPUs, and 16 GB memory. The machine learning models were trained using PyTorch 1.11.0 (Paszke et al., 2019), and implemented using PyTorchGeometric 2.0.4 (Fey and Lenssen, 2019).

4.1.1. Case study

In order to examine the output results of the proposed methods, we consider a representative test case where a chemist models a chemical property and tries to optimise it. This is a common use case of CAMD (Frühbeis et al., 1987). 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. Specifically, as an exemplar, we maximise the (normalised) boiling point of the (feasible) molecules. The boiling point $T_b(K)$ is calculated by $T_b(K) = \text{mean} + \text{std} \times \text{obj\_val}$. Note that our interest is to test the potential of the GNN modelling developed in Section 3, not to solve a physically-meaningful design problem. Models and training hyper-parameters were selected heuristically following some preliminary computer experiments discussed in Appendix E. Note also that accurately training GNNs is not the focus of this work.

The input space constraints were extended by including the possibility to find double and triple bonds, implemented using the input constraints described by Eqs. (25). Initial testing of the computer experiments found problems with steric constraints on the model, and to circumvent this, molecules with loops were excluded from the search space. We use the normal boiling point, i.e., at 1 atm).

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: (i) only single and double bonds allowed, (ii) only single and triple bonds allowed, and (iii) single, double, and triple bonds allowed. As double and triple bonds are handled using extra features, three different neural network configurations were trained. The GNNs are trained using the best hyperparameters found in our preliminary computer experiments, and the model 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 they are known molecules in public databases and chemical suppliers including PubChem (Kim et al., 2016), ChemSpider, synquestlabs, alfa aesar, matrix scientific.

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 Tb. 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 Tb 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).

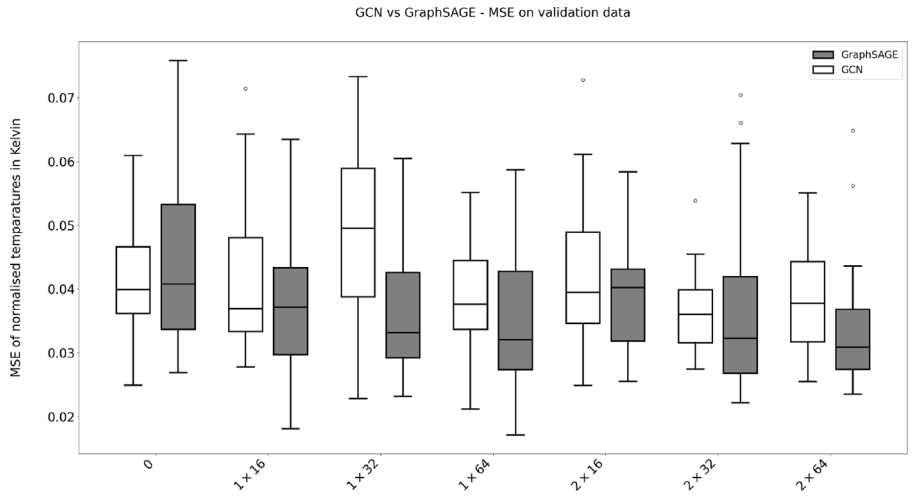


Fig. 2. Box-plots of the MSE of the validation data, for the GCN and GraphSAGE models for different layer depths and node width, after independently running the models 20 times for each configuration. The box-plots indicate the median, the lower and upper quartile and the lowest and largest MSE, when outliers, which are indicated by the dots, are excluded.

Table 2
Solving times (up to 36 000 s; smaller is better) and remaining optimality gaps (smaller is better) for the GCN MIQCP formulation for different molecule lengths.

	Num. layers	0	1			2		
	Nodes/layer	0	16	32	64	16	32	64
Molecule length 4	Time (s)	45	536	6426	13 404	9428	–	–
	MIP Gap	0.00	0.00	0.00	0.00	0.00	3.59	30.19
Molecule length 6	Time (s)	2309	–	–	–	–	–	–
	MIP Gap	0.00	2.92	8.61	8.20	31.61	47.10	93.86
Molecule length 8	Time (s)	–	–	–	–	–	–	–
	MIP Gap	0.54	7.79	14.21	12.11	39.32	63.47	104.51

4.2. Results

The following section describes the results that were found in the computer 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. Lastly, the CAMD case study is described, showing which molecules were found using the proposed methods.

4.2.1. Initial computer experiments

For both GCN and GraphSAGE models, we trained seven configurations (differing in the number of hidden layers and nodes per hidden layer) using the MSE as a loss function. A comparison of the box plots of the MSE of the models can be found in Fig. 2. Recall that temperatures are in normalised Kelvin.

For the GCN model, the minimum MSE values range from 0.0213 to 0.0278 and the median MSE values range from 0.0360 to 0.0495. For the GraphSAGE model the minimum MSE values range from 0.0171 to 0.0277 and the median from 0.0308 to 0.0438. In case of the GraphSAGE we can see that median MSE values decrease as the number of nodes per layer increase. For GCN no such pattern can be detected. Comparing the model accuracy of both GNN models shows 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 0.0171. However, the 1 x 16 is a close second with an MSE of 0.0181.

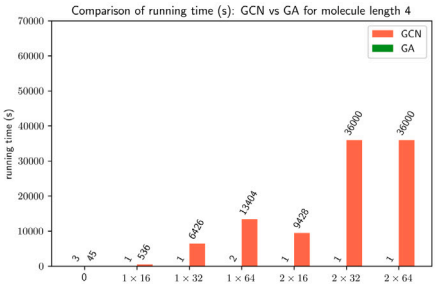
We then optimised the MILP and MIQCP formulations of the trained GNNs with a maximum time limit of 10 h. Table 2 compares the results for the optimisation of the MIQCP formulation for the different configurations. Only six out of the 21 experiments were solved within 10 h. All configurations were solved for molecule length $n = 4$ apart from 2 x 32 and 2 x 64, with solution times increasing with the number of nodes. For $n = 6$ only the formulation with 0 hidden layers was solved to optimality. 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. As the node depth increases, the remaining optimality gap after 10 h 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.

Table 3 shows the results of optimising the MILP formulations of the trained GraphSAGE neural networks. Only 5 configurations were solved to optimality. All the others terminated after a time limit of 10 h. Of the solved cases, four optima were found when the search space was limited to atoms of length 4 ($n = 4$), the other one was found when $n = 6$. Once again, the solution times increase with both the size of the GNN and the length of the molecules. We see that for $n = 4, 6$, as the node depth increases, the remaining 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 increases, the optimality gap also increases.

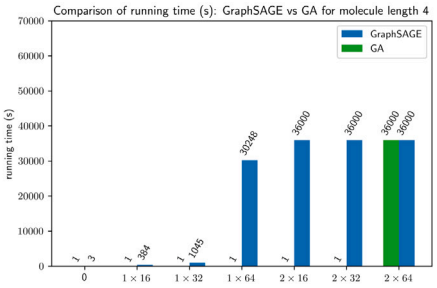
We can also compare between the GCN and GraphSAGE models. As mentioned previously, six of the GCN formulations were solved within 10 h, while only five of the GraphSAGE formulations were solved. For the solved problems, the GraphSAGE model is generally faster, apart from the configurations 1 x 64 and 2 x 16 for $n = 4$, where in the latter case GCN solves to optimality and GraphSAGE does not. We also observe that GraphSAGE has a better optimality gap than the GCN

Table 3
Solving times (up to 36 000 s; smaller is better) and remaining optimality gaps (smaller is better) for the GraphSAGE MIQCP formulation for different molecule lengths.

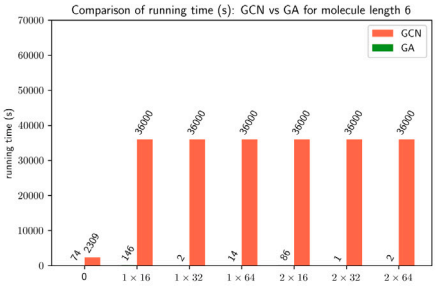
	Num. layers	0	1			2		
	Nodes/layer	0	16	32	64	16	32	64
Molecule length 4	Time (s)	3	384	1045	30 248	–	–	–
	MIP Gap	0.00	0.00	0.00	0.00	0.91	4.93	13.99
Molecule length 6	Time (s)	1108	–	–	–	–	–	–
	MIP Gap	0.00	0.57	2.29	8.49	5.57	11.39	17.26
Molecule length 8	Time (s)	–	–	–	–	–	–	–
	MIP Gap	0.33	5.19	3.63	11.18	7.11	13.60	26.54



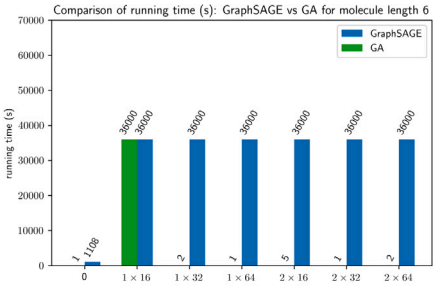
(a) GA vs GCN, $n = 4$



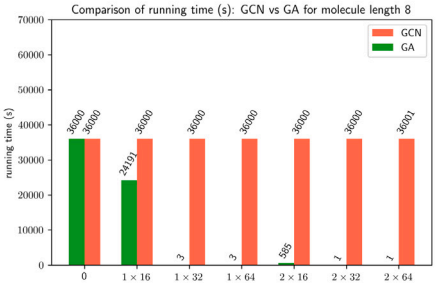
(b) GA vs GraphSAGE, $n = 4$



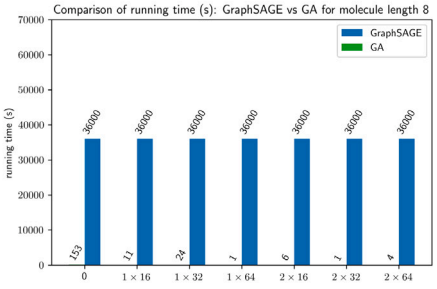
(c) GA vs GCN, $n = 6$



(d) GA vs GraphSAGE, $n = 6$



(e) GA vs GCN, $n = 8$



(f) GA vs GraphSAGE, $n = 8$

Fig. 3. 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.

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 a baseline. Fig. 3 shows this comparison. For 35 out of 42 instances (83%), the GA found an equally good solution as the deterministic optimiser in less than 2 min. For 3 instances, the GA did not find an equally good objective value and terminated because it reached the time limit of 10 hours.

4.2.2. 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 F.1. Recall from Section 4.1.1 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

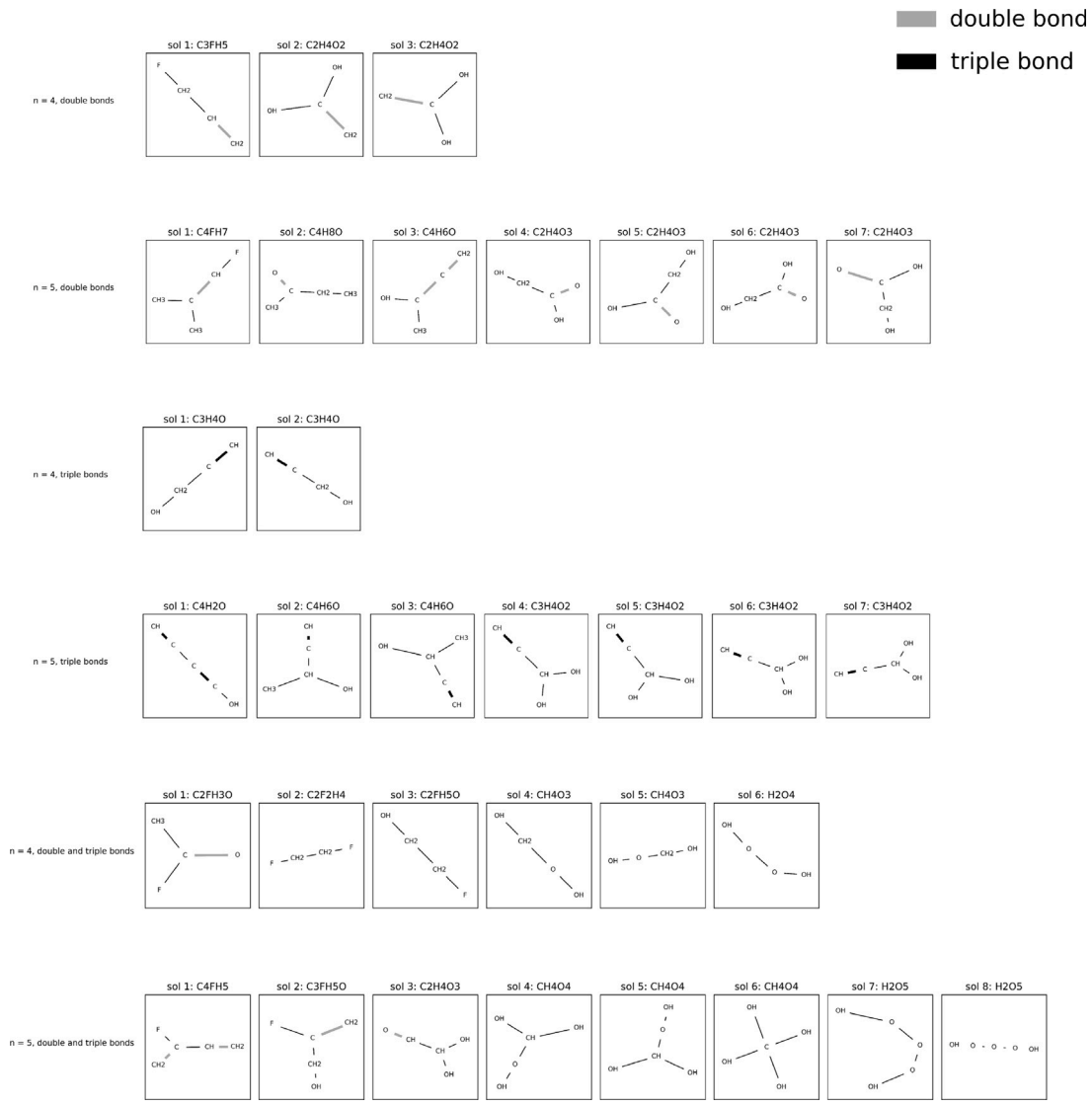


Fig. 4. The molecules associated with the best found lower bounds during the branch and bound optimisation process of the GraphSAGE formulation with 1 hidden layer and 16 nodes.

Table 4
MILP size for the 1 × 16 configuration of the GraphSAGE model.

Length	Search space	Constraints	Variables		
			Continuous	Integer	Total
4	Double bonds	4800	1345	540	1885
4	Triple bonds	4800	1345	540	1885
4	Double and triple bonds	4919	1345	576	1921
5	Double bonds	7254	1905	756	2661
5	Triple bonds	7254	1905	756	2661
5	Double and triple bonds	7438	1905	811	2716

single, double, and triple bonded molecules. Graphical representations of the found solutions for $n = 4$ and $n = 5$ are shown in Fig. 4. The size of the MILP is reported in Table 4.

There are multiple molecules obtained from the outcomes of the MILP solver. This was achieved using Gurobi's Solution Pool tool. Specifically, while the goal of a MILP solver is to find the globally optimal solution, other feasible solutions can be found as an algorithm progresses as an indirect output. We note that MILP can in this way produce a candidate pool similar to how GA can.

There are repeated molecules in the solution set. These are solutions that have different adjacency matrices but constitute to the same molecule. In total, there were 20 unique molecule-like structures found during the optimisation of the different search spaces. The analysis of these molecules can be found in Table 5. Zhang et al. (2023) develop symmetry-breaking constraints, which is a route to prevent generation of repeat molecules; their work applies to MILP and also to GA approaches, and could be added to our formulations in the future.

For 12 of the 20 molecules, we found literature sources indicating that the molecules were experimentally observed through a manual search of SMILES strings on public databases and chemical suppliers including PubChem (Kim et al., 2016), ChemSpider, synquestlabs, alfa aesar, matrix scientific. For the 12 observed molecules, 9 were synthesised, and their boiling points were recorded. Two of the molecules were not experimentally observed but were mentioned in papers as hypothetically possible under large pressure. Two of the molecules found during the optimisation process were also present in the training data set; all the others were not.

The absolute difference between the experimentally observed boiling points and the predicted boiling points from the optimisation ranged from 282.59 K to 345.39 K. The relative difference, calculated by the difference divided by the experimental boiling point, ranged from 2.7%–24.2%.

Table 5
Table with all computer experimental results and interpretation of the case study. Experimentally observed instances (in the literature) indicated with a star come from a database of a chemical supplier. The predicted T_b (K) is calculated by T_b (K) = mean + std× obj_val, mean = 312.64, std = 62.98, where the mean and std are from the training data set.

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[a]	253.15	Allyl fluoride	FALSE	0.02	313.17	60.02
		C2H4O2	TRUE (Mardyukov et al., 2020)		1,1-Dihydroxyethene	FALSE	0.78	361.61	
	5	C4FH7	FALSE		n/a	FALSE	0.27	329.79	
		C4H8O	TRUE[b]	343.15	Butanone	TRUE	0.63	352.59	9.44
		C4H6O	FALSE		n/a	FALSE	0.72	357.90	
		C2H4O3	TRUE[b]	375.15	Glycolic acid	FALSE	1.35	397.92	22.77
Triple bonded	4	C3H4O	TRUE[b]	377.15	Propargyl alcohol	FALSE	0.57	348.49	−28.66
	5	C4H2O	TRUE (Araki and Kuze, 2008)		Butadiynol	FALSE	0.19	324.31	
		C4H6O	TRUE[a]	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 & triple bonded	4	C2FH3O	TRUE[c]	283.15	Acetyl fluoride	FALSE	0.20	324.94	41.79
		C2F2H4	TRUE[a]	293.85	1,2-Difluorethane	TRUE	0.71	357.30	63.45
		C2FH5O	TRUE[b]	366.15	2-Fluoroethanol	FALSE	1.15	385.06	18.91
		CH4O3	FALSE		Hydroperoxymethanol	FALSE	1.63	415.34	
		H2O4	TRUE (Levanov et al., 2011)		Tetraoxidane	FALSE	1.80	425.95	
	5	C4FH5	FALSE		2-Fluoro-1,3-butadiene	FALSE	0.01	313.17	
		C3FH5O	TRUE[c]	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 (Böhm et al., 1997)[d]		Orthocarbonic acid	FALSE	2.33	459.23	
		H2O5	FALSE (Levanov et al., 2011)[d]		Pentaoxidane	FALSE	2.39	463.40	

[a] Matrix Scientific.
[b] Alfa Aesar.
[c] Synquest.
[d] Hypothetical molecule in cited paper.

5. Discussion

5.1. Model complexity

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

However, the number 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 number 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.

While theoretical comparisons between the models has value, we also consider our empirical findings.

5.2. Individual Computer 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. Some improvements 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 that are multiple orders of magnitude larger than the graphs that we currently find. This means that the proposed optimisation formulations cannot be used in other contexts where large graph neural networks are used, like road network modelling, or recommender systems. This implies that the proposed techniques, in its current formulation, should be used for small graph optimisations only, like molecule optimisation. At the same time, the encountered computational complexity also motivates the use of inexact solvers, such as the genetic algorithm.

Second, we note that the use of a deterministic optimiser (i.e., Gurobi for MILPs) has the advantage of knowing how close one is to the actual solution, while running the algorithm, expressed by the optimality gap. However, the computer 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 2×64 instance of the GraphSAGE model for 10 h. 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.

Third, we discuss the simple fact that when the number of nodes per layer increases, 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 number of layers and nodes increases, the number of decision variables and constraints increases, 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 increases linearly with the number 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.

5.3. Comparison of the computer experiments

As discussed above, the GraphSAGE model is generally better than the GCN model in terms of solving times; and when not solved to optimality, better also in terms of an optimality gap. There are instances where the GCN is better than the GraphSAGE model empirically. First we note that the bounds of the GCN models are smaller than those of the GraphSAGE model, for equal node depth and layer width. As mentioned before, smaller bounds result in faster solving times. However, that the bound difference is generally present for all instances when comparing the GCN and GraphSAGE, suggests there must be another reason for these exceptions.

Second, since training a neural network is a stochastic process, that training different neural networks with the same hyper-parameters does not result in the same weights and biases. We hypothesise that optimising different trained GNNs with the same configurations can result in different solving times. This is because 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 F.2. The same experiment for the instance 2×16 was repeated 5 times. The results shows that different trained neural network parameters result in different solving times when optimised using a deterministic solver. This confirms our hypothesis. Despite this, 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.

A final point pertaining to the initial computer experiments is the genetic algorithm baseline comparisons. It is clear from the results that in most cases the GA 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 main shortcoming of the GA: having no guarantee of convergence to the global optimum.

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

It is noteworthy that only two of the 20 molecules 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 generalisation ability points to a real-life use case for the proposed formulation in this article. However, this extrapolation over the training data domain can also contribute towards larger prediction errors (as observed in Table 5). In future works, we recommend modelling a validity domain and adding this as a constraint during optimisation (Schweidtmann et al., 2022).

There are two final remarks we would like to make on the found solutions, recognising that the modelling of the chemical properties was not the main focus of this work. First, on the one hand, we do see that when experimental results (in the literature) 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. Second, 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 cannot draw immediate conclusions from this. However, one hypothesis is that this might suggest is that when modelled molecules are at the higher end of the boiling point spectrum, then the errors of the GNNs become larger.

We end the discussion by remarking that the focus of this work is on the formulation of GNNs as optimisation models. In terms of the number of atoms used to build molecules, the scale of case study problems studied remains very small in the context of CAMD.

6. Conclusion and outlook

The success of computer aided molecular design and the ubiquity of neural networks lead to the question whether one can optimally search for molecular designs, constrained by certain properties, by making use of graph neural networks. A key barrier to using ‘traditional’, non-graph structured networks is that they struggle to learn from non-euclidean data, whereas molecules are naturally modelled as graph-like structures, motivating the use of GNNs.

Recognising recent progress on exact formulations of non-graph neural networks as mixed integer (non-)linear programs, this work therefore formulated trained GNNs as MILP programming formulations. These formulations can be used as surrogate models in optimisation problems. In particular, we treated two classes of GNNs: the frequently-used GCN and the contemporary GraphSAGE. We developed a formulation of GCN as a MIQCP, and of GraphSAGE as a MILP.

In terms of accuracy, we hypothesised that the GCN would reach better model accuracy with fewer hidden layers and nodes per layer than GraphSAGE, due to the GCN model’s more complex architecture. The results (Fig. 2), do not support this hypothesis. For four of the seven configurations (hidden layers $\times$ nodes) the GraphSAGE model has a better median validation MSE while running for 20 iterations, and for four of the seven configurations, the GraphSAGE model has a lower minimum validation error than the GCN model. Overall, with the hyper-parameters tested, we achieved similar model accuracy for both the GCN and GraphSAGE model, even with the same number of hidden layers and nodes per layer.

In terms of solving speed, we hypothesised was that the GraphSAGE MILP formulations would be faster than the GCN MIQCP formulations because linear solvers are faster than non-linear solvers (in this case, bi-linear). The results (Tables 2 and 3) find that the GraphSAGE model was generally faster (four of the six 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 three configurations (12 out of 15 early terminated cases) the GraphSAGE model had a smaller optimality gap than the GCN model. Overall, there is evidence to suggest the MILP formulation of the GraphSAGE model solves to optimality faster than the MIQCP 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.

Our final contribution was to apply the MI(N)LP formulations to a case study of optimising the boiling points of molecules. The case study successfully derived a set of optimal molecules, given constraints

on the design space. Of the 20 molecules derived, 12 were found were experimentally observed. Of the other eight, the literature notes two as hypothetical molecules. These were able to be synthesised under very high pressure or were an unstable molecule of molecular reaction. The remaining six molecules appear to be novel; their chemical feasibility in practice would be tested in vitro studies.

Our work opens up several prominent research directions. First, the models themselves have potential for improvement with stronger bound tightening techniques, and we think techniques for tightening MLP MILP formulations can be applied to GNN MILPs also. Going beyond feasibility-based bound tightening, optimisation-based bound tightening techniques (OBBT), and the combined technique of Wang et al. (2021). It will also be relevant to investigate comparing formulations using integer versus binary MILP formulations.

Second, using GNNs in CAMD, there is opportunity in increasing the training set size and using more of the learnable features, and reconsidering linearise structures and the input constraints. We underline that *training* of the GNNs was not the main focus of the current work, and that we applied our novel formulations to GNN as a case study. Third, when deterministic optimisation is not the main priority for researchers, our straightforward GA for optimising trained GNNs already shows promise.

CRediT authorship contribution statement

Tom McDonald: Writing – original draft, Visualization, Software, Methodology, Investigation. **Calvin Tsay:** Writing – review & editing, Supervision. **Artur M. Schweidtmann:** Writing – review & editing, Supervision, Methodology, Conceptualization. **Neil Yorke-Smith:** Writing – review & editing, Supervision, Project administration, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgements

We thank the anonymous reviewers for their useful suggestions on this work. Thanks to S. van der Laan.

Funding

This work was partially supported the EU Horizon 2020 research and development programme, grant number 952215 (TAILOR).

Appendix A. Linear formulations of ReLU neural networks

The first exact MILP formulations emulate the ReLU operator using binary activation variables and big-M formulations. Later works introduce tighter alternatives to big-M (Anderson et al., 2020; Tsay et al., 2021) or formulations that consider multiple nodes simultaneously (Singh et al., 2019). There have been various applications like neural network verification (Fischetti and Jo, 2018; Tjeng et al., 2017; Bunel et al., 2018; Dutta et al., 2018), counting linear regions in NNs (Serra et al., 2018), and lossless compression of NNs (Kumar et al., 2019; Serra et al., 2020). NNs can also be used as surrogate models 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 (Grimstad and Andersson, 2019; Di Martino et al., 2022; Kody et al., 2022; Yang et al., 2021). Further, we note that (MILP formulations of) NNs as surrogate could be implemented in a larger model, such as integrated process and material design.

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 (Vielma, 2015b). As a result, multiple research papers have sections dedicated on how to tighten said bounds (Cheng et al., 2017; Dutta et al., 2018; Tjeng et al., 2017; Fischetti and Jo, 2018; Grimstad and Andersson, 2019). The most basic and weakest among those is interval arithmetic, where the input bounds are propagated through the neural network (Cheng et al., 2017; Tjeng et al., 2017; Anderson et al., 2020). Other bound tightening techniques (BTTs) generate two Linear Programming (LP) problems for each neuron in which the bounds of these neurons are minimised and maximised. Tjeng et al. (2017) 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 (2018) find even tighter bounds by not relaxing the activation variable for a neuron, but this is even more computationally expensive. Wang et al. (2021) 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.

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 subsection describes how to linearise, for each hidden layer $k \in \{1, \dots, K-1\}$, the following MLP layer:

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

where $\sigma(\cdot) = \max\{0, \cdot\}$ 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. (A.1). This paper considers the linearisation by Fischetti and Jo (2018); this suffices for us to go on to linearise GNNs. 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

$$w^T y + b = x - s. \tag{A.2}$$

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:

$$\begin{cases} z = 0 & \text{then } s \leq 0 \\ z = 1 & \text{then } x \leq 0 \\ z \in \{0, 1\} \end{cases} \tag{A.3}$$

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 upper bound and lower bound on the possible values of x and s respectively. If the left hand side of Eq. (A.2) 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. (A.2) 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:

$$x_j^k \leq u_j^k z_j^k \tag{A.4a}$$

$$s_j^k \leq -l_j^k (1 - z_j^k) \tag{A.4b}$$

$$z_j^k \in \{0, 1\}. \tag{A.4c}$$

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.

$$W^K x^{K-1} + b^K = x_1^K \tag{A.5a}$$
$$W_j^k x^{k-1} + b_j^k = x_j^k - s_j^k \quad \forall k \in \{1,\dots,K-1\}, \forall j \in \{1,\dots,n_k\} \tag{A.5b}$$
$$x_j^k \le u_j^k z_j^k \quad \forall k \in \{1,\dots,K-1\}, \forall j \in \{1,\dots,n_k\} \tag{A.5c}$$
$$s_j^k \le -l_j^k (1 - z_j^k) \quad \forall k \in \{1,\dots,K-1\}, \forall j \in \{1,\dots,n_k\} \tag{A.5d}$$
$$x_j^k, s_j^k \ge 0 \quad \forall k \in \{1,\dots,K-1\}, \forall j \in \{1,\dots,n_k\} \tag{A.5e}$$
$$z_j^k \in \{0,1\} \quad \forall k \in \{1,\dots,K-1\}, \forall j \in \{1,\dots,n_k\} \tag{A.5f}$$
$$x^0 \in \Omega \tag{A.5g}$$

In this formulation, W_j^k is row j of the weight matrix of layer k, which naturally has the same dimension as the output 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 (2019), 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. Note this has no effect on the output, however.

Appendix B. MI(N)LP formulations of the GNNs

B.1. GCN

$$W^{K_{MLP}} x^{(K_{MLP}-1)} + b^K = x_1^K \tag{B.1a}$$
$$W_j^k x^{k-1} + b_j^k = x_j^k - s_j^k \quad \forall k \in \{1,\dots,K_{MLP}-1\}, \tag{B.1b}$$
$$\forall j \in \{1,\dots,n_k\} \tag{B.1c}$$
$$x_j^k \le U_j^k z_j^k \quad \forall k \in \{1,\dots,K_{MLP}-1\}, \tag{B.1d}$$
$$\forall j \in \{1,\dots,n_k\} \tag{B.1e}$$
$$s_j^k \le -L_j^k (1 - z_j^k) \quad \forall k \in \{1,\dots,K_{MLP}-1\}, \tag{B.1f}$$
$$\forall j \in \{1,\dots,n_k\} \tag{B.1g}$$
$$x_j^0 = \sum_i H_{ij}^K \quad j \in \{1,\dots,n_K\} \tag{B.1h}$$
$$\sum_l \hat{s}_{il} W_j^{kT} H_l^{(k-1)} = H_{ij}^k - S_{ij}^k \quad \forall k \in \{1,\dots,K\}, \tag{B.1i}$$
$$\forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{B.1j}$$
$$H_{ij}^k \le U_{ij}^k Z_{ij}^k \quad \forall k \in \{1,\dots,K\}, \tag{B.1k}$$
$$\forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{B.1l}$$
$$S_{ij}^k \le -L_{ij}^k (1 - Z_{ij}^k) \quad \forall k \in \{1,\dots,K\}, \tag{B.1m}$$
$$\forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{B.1n}$$
$$d_i^+ = \sum_j \tilde{A}_{ij} \quad \forall i \in \{1,\dots,N\} \tag{B.1o}$$
$$p_{il} = d_i^+ (d_{max}+1) + d_l^+ \quad \forall i,l \in \{1,\dots,N\} \tag{B.1p}$$
$$0 = p_{il} - 1c_1^{il} - 2c_2^{il} - \cdots - (d_{max}+1)^2 c_{(d_{max}+1)^2}^{il} \quad \forall i,l \in \{1,\dots,N\} \tag{B.1q}$$
$$1 = c_1^{il} + \cdots + c_{(d_{max}+1)^2}^{il} \quad \forall i,l \in \{1,\dots,N\} \tag{B.1r}$$
$$s_{il} = c_1^{il} g_1 + \cdots + c_{(d_{max}+1)^2}^{il} g_{(d_{max}+1)^2} \quad \forall i,l \in \{1,\dots,N\} \tag{B.1s}$$
$$s_{il} - M(1 - A_{il}) \le \hat{s}_{il} \le M(1 - A_{il}) + s_{il} \quad \forall i,l \in \{1,\dots,N\} \tag{B.1t}$$
$$-MA_{il} \le \hat{s}_{il} \le MA_{il} \quad \forall i,l \in \{1,\dots,N\} \tag{B.1u}$$
$$0 \le x_j^k, s_j^k \in \mathbb{R} \quad \forall k \in \{1,\dots,K_{MLP}\}, \tag{B.1v}$$
$$\forall j \in \{1,\dots,n_k\} \tag{B.1w}$$
$$z_j^k \in \{0,1\} \quad \forall k \in \{1,\dots,K_{MLP}-1\}, \forall j \in \{1,\dots,n_k\} \tag{B.1x}$$
$$0 \le H_{ij}^k, S_{ij}^k \in \mathbb{R} \quad \forall k \in \{1,\dots,K\}, \tag{B.1y}$$
$$\forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{B.1z}$$
$$Z_{ij}^k \in \{0,1\} \quad \forall k \in \{1,\dots,K\}, \tag{B.1aa}$$
$$\forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \tag{B.1ab}$$
$$\hat{s}_{il}, s_{il} \in \mathbb{R} \quad \forall i,l \in \{1,\dots,N\} \tag{B.1ac}$$
$$p^{il}, c^{il} \in \{0,1\} \quad \forall i,l \in \{1,\dots,d_{max}+1\} \tag{B.1ad}$$
$$d_i^+ \in \{0,1\} \quad \forall i \in \{1,\dots,d_{max}+1\} \tag{B.1ae}$$
$$A_{ij} \in \{0,1\} \quad \forall i,j \in \{1,\dots,N\} \tag{B.1af}$$
$$H^0, A \in \Omega \tag{B.1ag}$$

B.2. Graphsage

$$W^{K_{MLP}} x^{(K_{MLP}-1)} + b^K = x_1^{K_{MLP}} \tag{B.2a}$$
$$W_j^k x^{k-1} + b_j^k = x_j^k - s_j^k \quad \forall k \in \{1,\dots,K_{MLP}-1\}, \tag{B.2b}$$
$$\forall j \in \{1,\dots,n_k\} \tag{B.2c}$$
$$x_j^k \le U_j^k z_j^k \quad \forall k \in \{1,\dots,K_{MLP}-1\}, \tag{B.2d}$$

$$\forall j \in \{1,\dots,n_k\} \quad \text{(B.2e)}$$

$$s_j^k \le -L_j^k(1-z_j^k) \qquad \forall k \in \{1,\dots,K_{MLP}-1\}, \quad \text{(B.2f)}$$

$$\forall j \in \{1,\dots,n_k\} \quad \text{(B.2g)}$$

$$x_j^0 = \sum_i H_{ij}^K \qquad j \in \{1,\dots,n_K\} \quad \text{(B.2h)}$$

$$(\hat{W}_j^k)^T H_i^{(k-1)} + (\bar{W}_j^k)^T \times \sum_l b_{il}^{(k-1)} = H_{ij}^k - S_{ij}^k \qquad \forall k \in \{1,\dots,K\}, \quad \text{(B.2i)}$$

$$\forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \quad \text{(B.2j)}$$

$$H_{ij}^k \le U_{ij}^k Z_{ij}^k \qquad \forall k \in \{1,\dots,K\}, \quad \text{(B.2k)}$$

$$\forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \quad \text{(B.2l)}$$

$$S_{ij}^k \le -L_{ij}^k(1-Z_{ij}^k) \qquad \forall k \in \{1,\dots,K\}, \quad \text{(B.2m)}$$

$$\forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \quad \text{(B.2n)}$$

$$H_l^k - M(1-A_{il}) \le b_{il}^k \qquad \forall k \in \{0,\dots,K-1\}, \quad \text{(B.2o)}$$

$$\forall i,l \in \{1,\dots,N\} \quad \text{(B.2p)}$$

$$b_{il}^k \le H_l^k + M(1-A_{il}) \qquad \forall k \in \{0,\dots,K-1\}, \quad \text{(B.2q)}$$

$$\forall i,l \in \{1,\dots,N\} \quad \text{(B.2r)}$$

$$-M(A_{il}) \le b_{il}^k \le M(A_{il}) \qquad \forall k \in \{0,\dots,K-1\}, \quad \text{(B.2s)}$$

$$\forall i,l \in \{1,\dots,N\} \quad \text{(B.2t)}$$

$$0 \le x_j^k, s_j^k \in \mathbb{R} \qquad \forall k \in \{1,\dots,K_{MLP}\}, \quad \text{(B.2u)}$$

$$\forall j \in \{1,\dots,n_k\} \quad \text{(B.2v)}$$

$$z_j^k \in \{0,1\} \qquad \forall k \in \{1,\dots,K_{MLP}-1\}, \quad \text{(B.2w)}$$

$$\forall j \in \{1,\dots,n_k\} \quad \text{(B.2x)}$$

$$0 \le H_{ij}^k, S_{ij}^k \in \mathbb{R} \qquad \forall k \in \{1,\dots,K\}, \quad \text{(B.2y)}$$

$$\forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \quad \text{(B.2z)}$$

$$Z_{ij}^k \in \{0,1\} \qquad \forall k \in \{1,\dots,K\}, \quad \text{(B.2aa)}$$

$$\forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,n_k\} \quad \text{(B.2ab)}$$

$$b_{il}^k \in \mathbb{R}^{n_k} \qquad \forall k \in \{1,\dots,K\}, \quad \text{(B.2ac)}$$

$$\forall i,l \in \{1,\dots,N\} \quad \text{(B.2ad)}$$

$$A_{ij} \in \{0,1\} \qquad \forall i,j \in \{1,\dots,N\} \quad \text{(B.2ae)}$$

$$H^0, A \in \Omega \quad \text{(B.2af)}$$

Appendix C. Molecule properties background knowledge

Recall from Section 3.3 that 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 therefore 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 knowledge summarised in Table 1.

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

$$A_{11} = A_{22} = A_{12} = 1 \quad \text{(C.1a)}$$

$$A_{ii} \ge A_{(i+1),(i+i)} \qquad \forall i \in \{1,\dots,n-1\} \quad \text{(C.1b)}$$

$$A_{ii} = x_{i,1} + \cdots + x_{i,4} \qquad \forall i \in \{1,\dots,n\} \quad \text{(C.1c)}$$

$$A_{ii} = x_{i,5} + \cdots + x_{i,9} \qquad \forall i \in \{1,\dots,n\} \quad \text{(C.1d)}$$

$$A_{ii} = x_{i,10} + \cdots + x_{i,14} \qquad \forall i \in \{1,\dots,n\} \quad \text{(C.1e)}$$

$$4x_{i,1} + 2x_{i,2} + 1x_{i,3} + 1x_{i,4} = 0x_{i,5} + 1x_{i,6} + 2x_{i,7} + 3x_{i,8} + 4x_{i,9} + 0x_{i,10} + 1x_{i,11} + 2x_{i,12} + 3x_{i,13} + 4x_{i,14} \qquad \forall i \in \{1,\dots,n\} \quad \text{(C.1f)}$$

$$\sum_{j,i\ne j} A_{ij} = 0x_{i,5} + 1x_{i,6} + 2x_{i,7} + 3x_{i,8} + 4x_{i,9} \qquad \forall i \in \{1,\dots,n\} \quad \text{(C.1g)}$$

$$A_{ij} = A_{ji} \qquad \forall i,j \in \{1,\dots,n\} \quad \text{(C.1h)}$$

$$M_4 A_{ii} \ge \sum_{j,i\ne j} A_{ij} \qquad \forall i \in \{1,\dots,n\} \quad \text{(C.1i)}$$

$$A_{ii} \le \sum_{j,i\ne j} A_{ij} \qquad \forall i \in \{1,\dots,n\} \quad \text{(C.1j)}$$

$$M_5 A_{ii} \ge \sum_{f=1}^{14} x_{if} \qquad \forall i \in \{1,\dots,n\} \quad \text{(C.1k)}$$

$$A_{ii} \le \sum_{j<i} A_{ij} \qquad \forall i \in \{3,\dots,n\} \quad \text{(C.1l)}$$

where n is the number of atoms in the molecule, the big-M values are defined as $M_4 = n+1$ and $M_5 = |F| = 14$. Intuitively, the atom i is 'on' when $A_{ii} = 1$.

The intuition of the above constraints is as follows:

(a) Molecules must be at least length 2 and connected (see also constraint (*l*)).
(b) This is a symmetry braking constraint to enforce no gaps between activated atoms.
(c) Only one atom type.
(d) Only one number of neighbours to be indicated.
(e) Only one number of hydrogen neighbours to be indicated.
(f) The covalence of the atom must equal the sum of number of neighbours and the number of hydrogen neighbours.
(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.
(i) An atom is active if it is connected to others.
(j) An atom not connected to any others is deactivated.
(k) Feature vectors of atom i are 0 if $A_{ii} = 0$.
(l) No disconnected sub-graphs.

Appendix D. Genetic algorithm design

The purpose of the GA is to comprise a baseline non-exact method, in comparison to MI(N)LP, for optimising over trained GNN models. In this section we explain the operation of the GA.

D.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. The search space of molecules is given by constraints (C.1). 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 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.

The fitness function is defined by the GNNs, in the case of this paper, this is either the GCN model and the GraphSAGE model. The only exception is that there is a heavy penalty for unconnected graphs. Lastly, 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.

D.2. Crossover for GNNs for chemical property modelling

The crossover procedure in GAs is commonly 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.

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

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 as before.

There might be instances of an instance of offspring $(O1, OT1)$ which is not a feasible molecule. To make the molecule feasible, we undertake 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 are 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 (C.1).

D.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 is switched to another molecule.

Finally, the intrinsic termination condition of the algorithm is after τ seconds.

Appendix E. 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 obtain 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 (2017) and the GraphSAGE model by Hamilton et al. (2017). 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

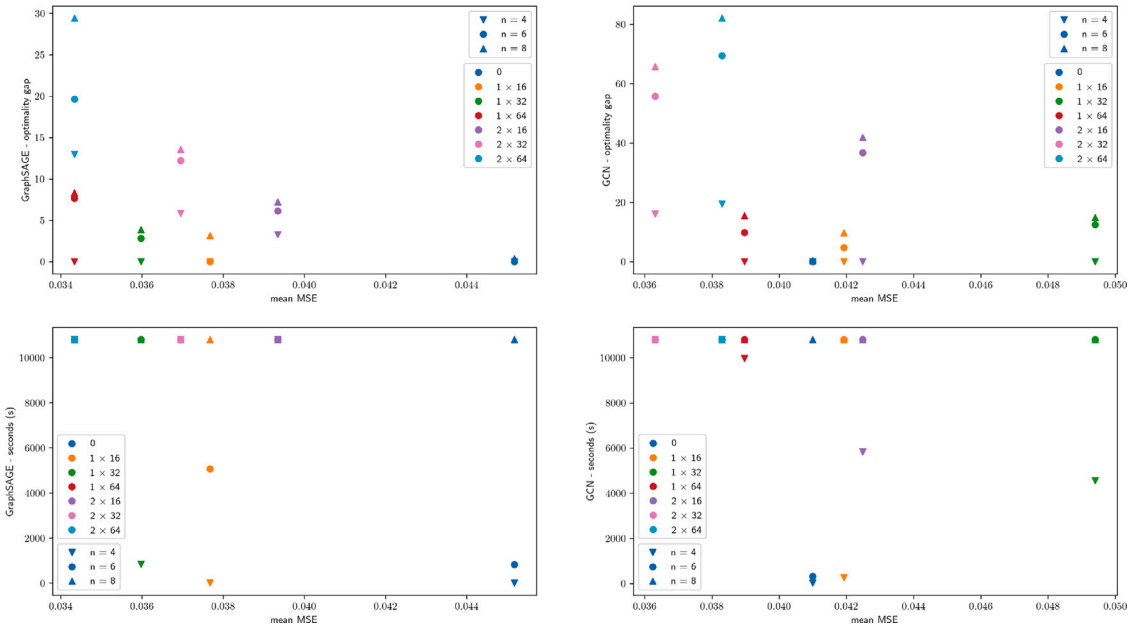


Fig. F.5. Model selection: trade-off between MSE versus solving time and optimality gap.

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, and set following Schweidtmann et al. (2020). We used the same hyper-parameters since the training of the GNNs in this paper was a simplified version of Schweidtmann et al.'s GNN model. 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 seven 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 seven 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 Appendices B.1 and B.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 h on the virtual machine. The experiments were compared on the solving time and optimality gap; 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 as described in Appendix D. 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.

Appendix F. Additional results

F.1. Deciding on the model for the case study

Fig. F.5 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. F.5 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 1×16 configuration find a decent trade off between solving time and model accuracy. While the solving time for the 0 configuration is better for GraphSAGE, its model accuracy is far worse.

F.2. GNN parameters and MILP solving time

A hypothesis is that the parameters found from training the GNN could be influential to 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

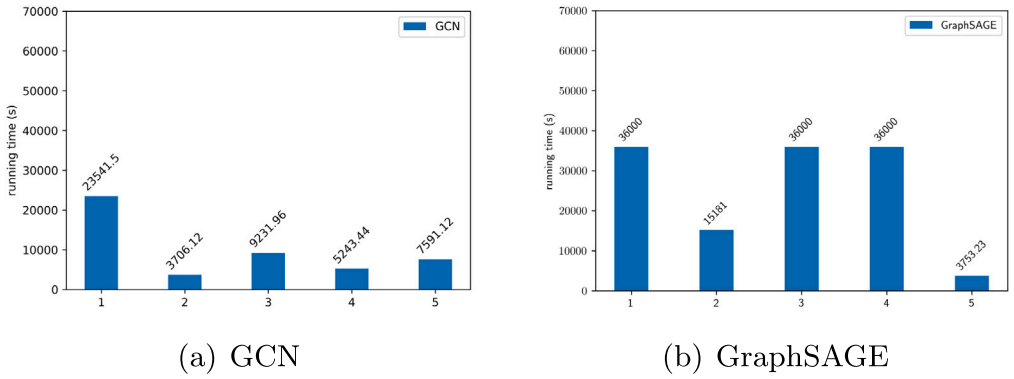


Fig. F.6. Bar graphs of the solving time of 5 randomly chosen trained models for both the GCN and GraphSAGE models, with node depth 16 and 2 hidden layers.

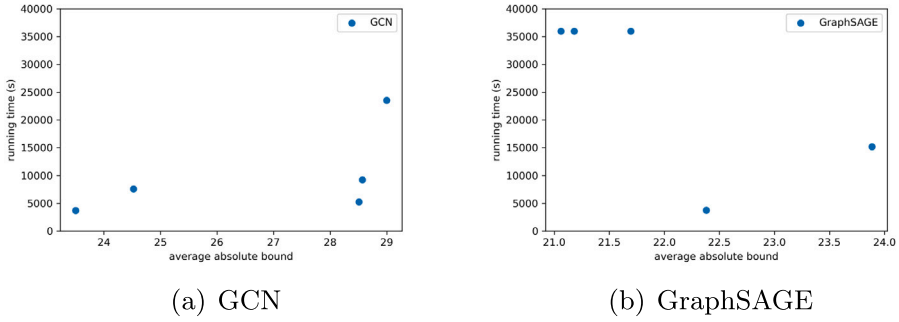


Fig. F.7. Scatter plots comparing the solving time and absolute average bound of 5 randomly chosen trained models for both the GCN and GraphSAGE models, with node depth 16 and 2 hidden layers.

the GCN model solved to optimality faster than the GraphSAGE model, which is unexpected. The five trained GNNs per model where randomly trained GNNs.

As can be seen in Fig. F.6, 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 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. F.7.

We find no correlation between the bounds and solving time for both models. Further investigation is thus warranted to see what factors are impactful on the solving time.

References

Achenie, L., Venkatasubramanian, V., Gani, R., 2002. Computer Aided Molecular Design: Theory and practice. Elsevier.

Alshehri, A.S., Gani, R., You, F., 2020. Deep learning and knowledge-based methods for computer-aided molecular design—toward a unified approach: State-of-the-art and future directions. Comput. Chem. Eng. 141, 107005.

Altae-Tran, H., Ramsundar, B., Pappu, A.S., Pande, V., 2017. Low data drug discovery with one-shot learning. ACS Cent. Sci. 3 (4), 283–293.

Anderson, R., Huchette, J., Ma, W., Tjandraatmadja, C., Vielma, J.P., 2020. Strong mixed-integer programming formulations for trained neural networks. Math. Program. 183 (1), 3–39.

Araki, M., Kuze, N., 2008. Laboratory detection of a linear carbon chain alcohol: HC4OH and its deuterated species. Astrophys. J. 680 (1), L93.

Austin, N.D., Sahinidis, N.V., Trahan, D.W., 2016. Computer-aided molecular design: An introduction and review of tools, applications, and solution techniques. Chem. Eng. Res. Des. 116, 2–26.

Bardow, A., Steur, K., Gross, J., 2010. Continuous-molecular targeting for integrated solvent and process design. Ind. Eng. Chem. Res. 49 (6), 2834–2840.

Battaglia, P.W., Hamrick, J.B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al., 2018. Relational inductive biases, deep learning, and graph networks. arXiv preprint arXiv:1806.01261.

Begam, B.F., Kumar, J.S., 2016. Computer assisted QSAR/QSPR approaches – A review. Indian J. Sci. Technol. 9 (8), 1–8.

Böhm, S., Antipova, D., Kuthan, J., 1997. A study of methanetetraol dehydration to carbonic acid. Int. J. Quantum Chem. 62 (3), 315–322.

Bouritsas, G., Frasca, F., Zafeiriou, S.P., Bronstein, M., 2022. Improving graph neural network expressivity via subgraph isomorphism counting. IEEE Trans. Pattern Anal. Mach. Intell..

Bunel, R., Turkaslan, I., Torr, P.H.S., Kohli, P., Mudigonda, P.K., 2018. A unified view of piecewise linear neural network verification. In: Advances in Neural Information Processing Systems. Vol. 31, pp. 4795–4804.

Cheng, C.-H., Nührenberg, G., Ruess, H., 2017. Maximum resilience of artificial neural networks. In: Proc. of International Symposium on Automated Technology for Verification and Analysis. pp. 251–268.

de Lima Ribeiro, F.A., Ferreira, M.M.C., 2003. QSPR models of boiling point, octanol–water partition coefficient and retention time index of polycyclic aromatic hydrocarbons. J. Mol. Struct. 663 (1–3), 109–126.

Defferrard, M., Bresson, X., Vandergheynst, P., 2016. Convolutional neural networks on graphs with fast localized spectral filtering. Adv. Neural Inf. Process. Syst. 29.

Di Martino, M., Avraamidou, S., Pistikopoulos, E.N., 2022. A neural network based superstructure optimization approach to reverse osmosis desalination plants. Membranes 12 (2), 199.

Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A., 2018. Output range analysis for deep feedforward neural networks. In: Proc. of 10th NASA Formal Methods Symposium. NFM'18, pp. 121–138. http://dx.doi.org/10.1007/978-3-319-77935-5_9.

Duvenaud, D.K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., Adams, R.P., 2015. Convolutional networks on graphs for learning molecular fingerprints. Adv. Neural Inf. Process. Syst. 28.

Egolf, L.M., Wessel, M.D., Jurs, P.C., 1994. Prediction of boiling points and critical temperatures of industrially important organic compounds from molecular structure. J. Chem. Inf. Comput. Sci. 34 (4), 947–956.

Errica, F., Podda, M., Bacciu, D., Micheli, A., 2019. A fair comparison of graph neural networks for graph classification. arXiv preprint arXiv:1912.09893.

Fey, M., Lenssen, J.E., 2019. Fast graph representation learning with PyTorch geometric. arXiv preprint arXiv:1903.02428.

Fischetti, M., Jo, J., 2018. Deep neural networks and mixed integer linear optimization. Constraints 23 (3), 296–309.

Folić, M., Adjiman, C.S., Pistikopoulos, E.N., 2007. Design of solvents for optimal reaction rate constants. AIChE J. 53 (5), 1240–1256.

Frühbeis, H., Klein, R., Wallmeier, H., 1987. Computer-assisted molecular design (CAMD)—An overview. Angew. Chem. Int. Ed. Engl. 26 (5), 403–418.

Gani, R., 2019. Group contribution-based property estimation methods: advances and perspectives. Curr. Opin. Chem. Eng. 23, 184–196.
Gani, R., Nielsen, B., Fredenslund, A., 1991. A group contribution approach to computer-aided molecular design. AIChE J. 37 (9), 1318–1332.
Gao, H., Wang, Z., Ji, S., 2018. Large-scale learnable graph convolutional networks. In: Proc. of 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 1416–1424.
Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E., 2017. Neural message passing for quantum chemistry. In: Proc. of International Conference on Machine Learning. ICML'17, PMLR, pp. 1263–1272.
Gleixner, A.M., Berthold, T., Müller, B., Weltge, S., 2017. Three enhancements for optimization-based bound tightening. J. Global Optim. 67 (4), 731–757.
Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press, http://www.deeplearningbook.org.
Grimstad, B., Andersson, H., 2019. Relu networks as surrogate models in mixed-integer linear programs. Comput. Chem. Eng. 131, 106580.
Gurobi Optimization, 2022. Gurobi optimizer reference manualversion 9.5.1. URL www.gurobi.com.
Ha, Z., Ring, Z., Liu, S., 2005. Quantitative structure- property relationship (QSPR) models for boiling points, specific gravities, and refraction indices of hydrocarbons. Energy Fuels 19 (1), 152–163.
Hamilton, W., Ying, Z., Leskovec, J., 2017. Inductive representation learning on large graphs. Adv. Neural Inf. Process. Syst. 30.
Henaff, M., Bruna, J., LeCun, Y., 2015. Deep convolutional networks on graph-structured data. arXiv preprint arXiv:1506.05163.
Hilal, S., Karickhoff, S., Carreira, L., 2003. Prediction of the vapor pressure boiling point, heat of vaporization and diffusion coefficient of organic compounds. QSAR Combin. Sci. 22 (6), 565–574.
Hu, W., Liu, B., Gomes, J., Zitnik, M., Liang, P., Pande, V., Leskovec, J., 2019. Strategies for pre-training graph neural networks. arXiv preprint arXiv:1905.12265.
Huchette, J., Muñoz, G., Serra, T., Tsay, C., 2023. When deep learning meets polyhedral theory: A survey. arXiv preprint arXiv:2305.00241.
Ivanciuc, O., Ivanciuc, T., Balaban, A.T., 2002. Quantitative structure–Property relationships for the normal boiling temperatures of acyclic carbonyl compounds. Internet Electron. J. Mol. Des. 1, 252–268.
Katritzky, A.R., Lobanov, V.S., Karelson, M., 1995. QSPR: the correlation and quantitative prediction of chemical and physical properties from structure. Chem. Soc. Rev. 24 (4), 279–287.
Kim, S., Thiessen, P.A., Bolton, E.E., Chen, J., Fu, G., Gindulyte, A., Han, L., He, J., He, S., Shoemaker, B.A., et al., 2016. PubChem substance and compound databases. Nucleic Acids Res. 44 (D1), D1202–D1213.
Kipf, T.N., Welling, M., 2016. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.
Kipf, T.N., Welling, M., 2017. Semi-supervised classification with graph convolutional networks. In: Proc. of 5th International Conference on Learning Representations (ICLR'17). URL https://openreview.net/forum?id=SJU4ayYgl.
Kody, A., Chevalier, S., Chatzivasileiadis, S., Molzahn, D., 2022. Modeling the AC power flow equations with optimally compact neural networks: Application to unit commitment. Electr. Power Syst. Res. 213, 108282.
Kumar, A., Serra, T., Ramalingam, S., 2019. Equivalent and approximate transformations of deep neural networks. arXiv preprint arXiv:1905.11428.
Levanov, A.V., Sakharov, D.V., Dashkova, A.V., Antipenko, E.E., Lunin, V.V., 2011. Synthesis of hydrogen polyoxides H2O4 and H2O3 and their characterization by Raman spectroscopy. Eur. J. Inorg. Chem. 33, 5144–5150.
Liao, R., Zhao, Z., Urtasun, R., Zemel, R.S., 2019. LanczosNet: Multi-scale deep graph convolutional networks. In: Proc. of 7th International Conference on Learning Representations. ICLR'19.
Lusci, A., Pollastri, G., Baldi, P., 2013. Deep architectures and deep learning in chemoinformatics: the prediction of aqueous solubility for drug-like molecules. J. Chem. Inf. Model. 53 (7), 1563–1575.
Mansimov, E., Mahmood, O., Kang, S., Cho, K., 2019. Molecular geometry prediction using a deep generative graph neural network. Sci. Rep. 9 (1), 1–13.
Mardyukov, A., Eckhardt, A.K., Schreiner, P.R., 2020. 1,1-Ethenediol: The long elusive enol of acetic acid. Angew. Chem. Int. Ed. 59 (14), 5577–5580.
McDonald, T., 2022. Mixed Integer (Non-) Linear Programming Formulations of Graph Neural Networks (Master's thesis). Delft University of Technology.
Micheli, A., 2009. Neural network for graphs: A contextual constructive approach. IEEE Trans. Neural Netw. 20 (3), 498–511.
Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., Bronstein, M.M., 2017. Geometric deep learning on graphs and manifolds using mixture model cnns. In: Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'17). pp. 5115–5124.
Niepert, M., Ahmed, M., Kutzkov, K., 2016. Learning convolutional neural networks for graphs. In: International Conference on Machine Learning. ICML'16, PMLR, pp. 2014–2023.
Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S., 2019. PyTorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems. Vol. 32, pp. 8024–8035, URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.
Razakh, T.M., Wang, B., Jackson, S., Kalia, R.K., Nakano, A., Nomura, K., Vashishta, P., 2021. PND: Physics-informed neural-network software for molecular dynamics applications. SoftwareX 15, 100789. http://dx.doi.org/10.1016/J.SOFTX.2021.100789.
Rittig, J.G., Gao, Q., Dahmen, M., Mitsos, A., Schweidtmann, A.M., 2022a. Graph neural networks for the prediction of molecular structure-property relationships. arXiv preprint arXiv:2208.04852.
Rittig, J.G., Ritzert, M., Schweidtmann, A.M., Winkler, S., Weber, J.M., Morsch, P., Heufer, K.A., Grohe, M., Mitsos, A., Dahmen, M., 2022b. Graph machine learning for design of high-octane fuels. AIChE J. 69 (4), e17971. http://dx.doi.org/10.1002/aic.17971.
Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G., 2008. The graph neural network model. IEEE Trans. Neural Netw. 20 (1), 61–80.
Schweidtmann, A.M., Mitsos, A., 2019. Deterministic global optimization with artificial neural networks embedded. J. Optim. Theory Appl. 180 (3), 925–948.
Schweidtmann, A.M., Rittig, J.G., König, A., Grohe, M., Mitsos, A., Dahmen, M., 2020. Graph neural networks for prediction of fuel ignition quality. Energy Fuels 34 (9), 11395–11407.
Schweidtmann, A.M., Weber, J.M., Wende, C., Netze, L., Mitsos, A., 2022. Obey validity limits of data-driven models through topological data analysis and one-class classification. Opt. Eng. 23 (2), 855–876.
Serra, T., Kumar, A., Ramalingam, S., 2020. Lossless compression of deep neural networks. In: Proc. of International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR'20). pp. 417–430.
Serra, T., Tjandraatmadja, C., Ramalingam, S., 2018. Bounding and counting linear regions of deep neural networks. In: International Conference on Machine Learning. PMLR, pp. 4558–4566.
Singh, G., Ganvir, R., Püschel, M., Vechev, M., 2019. Beyond the single neuron convex barrier for neural network certification. Adv. Neural Inf. Process. Syst. 32.
Stops, L., Leenhouts, R., Gao, Q., Schweidtmann, A.M., 2023. Flowsheet generation through hierarchical reinforcement learning and graph neural networks. AIChE J. 69 (1), e17938.
Tjeng, V., Xiao, K., Tedrake, R., 2017. Evaluating robustness of neural networks with mixed integer programming. arXiv preprint arXiv:1711.07356.
Tsay, C., Kronqvist, J., Thebelt, A., Misener, R., 2021. Partition-based formulations for mixed-integer optimization of trained ReLU neural networks. Adv. Neural Inf. Process. Syst. 34, 3068–3080.
Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y., 2018. Graph attention networks. In: Proc. of 6th International Conference on Learning Representations. ICLR'18, URL https://openreview.net/forum?id=rJXMpikCZ.
Vielma, J., 2015a. Mixed integer linear programming formulation techniques. SIAM Rev. 57, http://dx.doi.org/10.1137/130915303.
Vielma, J.P., 2015b. Mixed integer linear programming formulation techniques. SIAM Rev. 57 (1), 3–57.
Wang, K., Lozano, L., Cardonha, C., Bergman, D., 2021. Acceleration techniques for optimization over trained neural network ensembles. arXiv preprint arXiv:2112.07007.
Wessel, M.D., Jurs, P.C., 1995. Prediction of normal boiling points of hydrocarbons from molecular structure. J. Chem. Inf. Comput. Sci. 35 (1), 68–76.
Wieder, O., Kohlbacher, S., Kuenemann, M., Garon, A., Ducrot, P., Seidel, T., Langer, T., 2020. A compact review of molecular property prediction with graph neural networks. Drug Discov. Today: Technol. 37, 1–12.
Withnall, M., Lindelöf, E., Engkvist, O., Chen, H., 2020. Building attention and edge message passing neural networks for bioactivity and physical–chemical property prediction. J. Cheminform. 12 (1), 1–18.
Wolsey, L., 2020. Integer Programming, seond ed. John Wiley & Sons, ISBN: 9781119606475.
Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Philip, S.Y., 2020. A comprehensive survey on graph neural networks. IEEE Trans. Neural Netw. Learn. Syst. 32 (1), 4–24.
Yang, S.-B., Li, Z., Wu, W., 2021. Data-driven process optimization considering surrogate model prediction uncertainty: A mixture density network-based approach. Ind. Eng. Chem. Res. 60 (5), 2206–2222.
Zhang, L., Cignitti, S., Gani, R., 2015. Generic mathematical programming formulation and solution for computer-aided molecular design. Comput. Chem. Eng. 78, 79–84.
Zhang, S., Salazar, J.S.C., Feldmann, C., Walz, D., Sandfort, F., Mathea, M., Tsay, C., Misener, R., 2023. Optimizing over trained GNNs via symmetry breaking. arXiv:2305.09420.
Zhang, J., Shi, X., Xie, J., Ma, H., King, I., Yeung, D., 2018. GaAN: Gated attention networks for learning on large and spatiotemporal graphs. In: Globerson, A., Silva, R. (Eds.), Proc. of 34th Conference on Uncertainty in Artificial Intelligence (UAI'18). AUAI Press, pp. 339–349, URL http://auai.org/uai2018/proceedings/papers/139.pdf.