EURO Journal on Computational Optimization 13 (2025) 100116

Contents lists available at ScienceDirect

EURO Journal on Computational Optimization

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

Unsupervised learning with GNNs for QUBO-based combinatorial optimization

Olga Krylova[a, b], Frank Phillipson[a, c,*]

[a] TNO, the Netherlands Organisation for Applied Scientific Research, Anna van Buerenplein 1, The Hague, 2595 DA, the Netherlands
[b] VU University Amsterdam, Faculty of Sciences, De Boelelaan 1111, Amsterdam, 1081 HV, the Netherlands
[c] Maastricht University, School of Business and Economics, Tongersestraat 53, Maastricht, 6211 LM, the Netherlands

HIGHLIGHTS

- Previously suggested GNN solver is effective on sparse graphs only.
- Replacing Adam with RPROP reduces solution times.
- GraphSAGE can handle denser graphs, while GCN performs better on sparse graphs.
- Transfer learning between MaxCut and MIS problems improves solver performance.

ARTICLE INFO

Keywords:
Unsupervised learning
GNNs
QUBO
Maximum cut
Maximum independent set

ABSTRACT

Recent advances in deep learning techniques pose a question of whether they can facilitate the task of finding good quality solutions to combinatorial optimization (CO) problems in a practically relevant solution time. Specifically, it is of practical relevance to determine to what extent graph neural networks (GNNs) can be applied to CO problems that can be formulated as QUBOs and thus be naturally interpreted as graph problems. In this research a GNN solver is applied to two classical CO problems–the maximum cut problem and maximum independent set problem–in an unsupervised learning setting. We show that while GNN solver consistently finds good quality solutions for the Max Cut problem irrespective of the size and density of the graph, solving MIS problems is challenging for all but very sparse graphs. We further show how this problem can be addressed by embedding transfer between these two problems and compare two different GNN architectures–GCN and GraphSAGE on their robustness with respect to graph density and symmetry. Finally we demonstrate that changing the widely used Adam optimizer to Rprop optimizer can lead to considerable reduction in solution times.

1. Introduction

Combinatorial optimization (CO) problems arise naturally in real life and finding good quality solutions to them as fast as possible is paramount for efficient operation of many businesses, processes and sectors such as transportation, financial and banking sectors, supply chains, electrical and traffic networks. Yet, a significant number of CO problems are notoriously difficult. In fact, in terms of computational complexity theory, many of them are known to be *NP-hard* [24]. NP-hardness boils down to the fact that finding optimal solutions to the real-life sized problems is often not feasible within practically relevant solution times. Nevertheless, since day-to-day existence and activities are often reliant on the solutions to such problems, they are being solved in practice by heuristics

* Corresponding author.
Email address: frank.phillipson@tno.nl (F. Phillipson).

https://doi.org/10.1016/j.ejco.2025.100116

Available online 27 September 2025

and approximation algorithms. Neither of these is guaranteed to deliver an optimal solution, however again, in practice having a solution of a reasonable quality within a feasible solution time is enough. In recent years, first machine learning and then deep learning approaches specifically started to be applied to the CO problems with the aim of delivering fast and reliable solutions to NP-hard problems. For a comprehensive review of use of ML approaches for CO problems see, e.g., [24,39].

Many CO problems are graph-related. Examples of such problems are the *maximal clique* problem or *the minimal vertex cover* problem. Some problems, although originally not formulated on graphs, can also be formulated as graph problems, e.g., the *hitting set problem* [9]. And even more generally, many CO problems, that can be formulated as Quadratic Unconstrained Binary Optimization (QUBO) problems, can be viewed as a graph, where variables represent graph vertices and interactions between those variables represent edges. Despite the fact that graph structures arise naturally in CO, for a long time graphs as an unstructured data structure posed a challenge for conventional machine learning methods and neural networks specifically. The emergence of *graph neural networks* (GNN) architectures that are capable of working directly with graphs as input stimulated a wave of new approaches and applications of deep learning to CO problems. Another way in which QUBO formulations facilitate the application of deep learning methods to CO problems is by providing unconstrained loss functions to the problems that might be initially constrained. Finally, use of binary decision variables offers a potentially straightforward relaxation of the problem to continuous probability variables, thus transforming discrete cost function into continuous loss required for machine learning methods.

From the machine learning perspective, CO problems can be approached within three main learning paradigms: supervised learning, unsupervised learning, and reinforcement learning. Supervised learning with GNNs with respect to CO problems poses a considerable challenge at the moment, since it requires a (huge) set of solved CO instances, which is difficult to obtain due to their NP-hardness. Nevertheless, there is a considerable amount of research into the topic (see, e.g., [25,28,32]). The reinforcement learning paradigm is a very promising research direction for CO problems since on the one hand it does not require a set of already solved NP-hard problems and on the other hand it can mimic existing algorithms and heuristics that often construct a solution iteratively. Examples of using GNNs within the reinforcement learning paradigm with respect to CO problems are the works of Khali et al. [19] and Kool et al. [23]. Finally, unsupervised learning potentially provides the simplest and the most straightforward approach to solving CO problems with GNNs, which does not require a presolved training set of problems and uses the provided cost function as a loss function to be minimized in the process of a network training.

However, previous research highlighted several challenges with respect to such applications. Firstly, ML requires continuous loss, but combinatorial problems are inherently discrete. A straightforward continuous relaxation of discrete cost functions offers on the one hand a way to overcome this obstacle but suffers from drawbacks that were pointed out by many authors previously, namely the lack of useful gradient information. This problem was investigated in the research of Donti et al. [11], Amos and Kolter [2], Vlastelica et al. [40] and Mandi and Guns [29] to name a few. Secondly, it was demonstrated that GNNs suffer from limited expressivity or representation ability. This was studied by Morris et al. [31] and Garg et al. [13]. It was shown that graph representations produced by GNNs might not be discriminative enough to guide the solution process. This problem is dependent among others on the maximal degree of the graph [8].

Despite the fact that these limitations and potential hurdles are known, most of the research that studies application of the GNNs to CO problems in an unsupervised manner do not specifically pay attention to how these issues might impact the reported results.

In this research, we aim to study in more detail an application of GNNs to CO problems in QUBO formulation.

We investigate through a series of experiments the extent to which GNN solver is applicable to the graphs of different sizes, densities and symmetries for finding good quality solutions within reasonable time budget for two well researched optimization problems the Maximum Cut Problem and the Maximum Independent Set Problem. We pay special attention to the deterioration of the solution quality with the growing degree of the graph, since inability of a GNN solver to solve MIS on denser graphs is one of the deficiencies of the GNN solver that was the main point of criticism in previous research (see Section 2.2). Moreover we apply several several machine learning methods and techniques to increase the GNN solver's robustness against growing density of the problem graph and map experimentally improvement in the solution quality that such techniques could deliver for MIS specifically. Namely, we compare two GNN architectures–GCN and GraphSAGE, and demonstrate how the choice of the optimizer can enhance the solution process on denser graphs as well as the solution times and investigate if embedding transfer between two problems can boost the GNN solver's ability to find good solutions for denser graphs.

This paper is structured as follows. Section 2 outlines the recent advances in the topic and provides more information about the concepts relevant to this research, such as the definition of the QUBO formulation and Graph Neural Network architectures. Section 3 is dedicated to the description of the experiments and the experimental setup with their results being presented in Section 4. Finally, Section 5 summarizes the results, discusses limitations of the study and offers avenues for the potential further research.

2. Background and previous research

In this section we give definitions of a QUBO, MaxCut and MIS problem as well as introduce their QUBO formulations. We further provide some information about Graph Neural Networks and optimizers, as well as discuss recent relevant advances on the topic of applying GNNs to the CO problems.

2.1. Background

2.1.1. QUBO

As was demonstrated in recent years [4,15,22] a lot of CO problems can be formulated as QUBO problems. The QUBO problem has the following form:

$$\min \quad \boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x} \tag{1a}$$

$$\text{s.t.} \quad \boldsymbol{x} \in \{0,1\}^n, \tag{1b}$$

where $\boldsymbol{x}$ is a vector of n binary decision variables and $\boldsymbol{Q} \in \mathbb{R}$ is an nxn upper triangular or symmetric matrix.

A QUBO formulation has several benefits. For this particular research, we note, first, that unconstrained cost functions, provided by the QUBO formulation, can act as a suitable loss function for deep learning applications. Secondly, as was mentioned in previous research (see e.g., [33]) QUBO defines a graph with vertex set $\boldsymbol{V}$, where $\boldsymbol{V} = \{x_1, x_2, \dots, x_n\}$, and weighted adjacency matrix $\boldsymbol{Q}$ is defined by the interactions/adjacency of the variables within the cost function and associated weights. This graph view translates the QUBO formulation to a data structure to which GNNs can be applied.

2.1.2. Maximum cut problem

Maximum Cut (MaxCut) is a classical CO problem defined as follows. Given an undirected graph $\boldsymbol{G}$ with a set of nodes $\boldsymbol{V}$ and a set of edges $\boldsymbol{E}$, the partitioning of $\boldsymbol{V}$ into two complementary subsets $\boldsymbol{V}_1$ and $\boldsymbol{V}_2$ is being searched for, such that the number of edges between those subsets of nodes is maximal. As shown in [15] MaxCut has the following QUBO formulation. Let $\boldsymbol{x} \in \{0,1\}^{|\boldsymbol{V}|}$, where x_i is a binary variable indicating whether a $v_i \in \boldsymbol{V}_1$ then

$$\max y = \sum_{i,j \in E} (x_i + x_j - 2x_i x_j), \tag{2}$$

The formulation is equivalent to the following minimization version of the cost function:

$$\min y = \sum_{i<j} \boldsymbol{A}_{ij} (2x_i x_j - x_i - x_j), \tag{3}$$

where $\boldsymbol{A}$ is an adjacency matrix of $\boldsymbol{G}$ or alternatively it can be rewritten (see [30]) as

$$\min y = \boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x}, \tag{4}$$

where $\boldsymbol{Q} = \boldsymbol{A} - \boldsymbol{D}$ and $\boldsymbol{D}$ is a degree matrix of $\boldsymbol{G}$. Interestingly, $\boldsymbol{Q} = -\boldsymbol{L}$, where $\boldsymbol{L}$ is a Laplacian matrix of $\boldsymbol{G}$.

2.1.3. Maximum independent set problem

Maximum Independent Set is another classical CO problem. Given a graph $\boldsymbol{G}$ with vertex set $\boldsymbol{V}$ and edge set $\boldsymbol{E}$, a subset $\boldsymbol{S}$ of $\boldsymbol{V}$ is being searched for such that no two vertices of $\boldsymbol{S}$ are adjacent in $\boldsymbol{G}$ and $\boldsymbol{G}$ has no independent set $\boldsymbol{S}'$ with $|\boldsymbol{S}'| > |\boldsymbol{S}|$. The classical formulation of MIS is that of a constrained optimization problem. Let $\boldsymbol{x} \in \{0,1\}^{|\boldsymbol{V}|}$, then

$$\max y = \sum_{i \in V} x_i \tag{5a}$$

$$\text{s.t. } x_i + x_j \leq 1, \quad \forall (i,j) \in \boldsymbol{E} \tag{5b}$$

Following [15], constraint (Eq. (5a)) can be incorporated into the cost function as the penalty term. Changing the problem from maximization to minimization, we obtain:

$$\min y = -\sum_{i \in V} x_i + P \sum_{(i,j) \in E} x_i x_j, \tag{6}$$

which is equivalent to:

$$\min y = \boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x}, \tag{7}$$

where $\boldsymbol{Q} = P\boldsymbol{A} - \boldsymbol{I}$ and $\boldsymbol{I}$ is a $|\boldsymbol{V}| \times |\boldsymbol{V}|$ identity matrix. Penalty weight P can be treated as a hyperparameter and optimized separately.

2.1.4. Graph neural networks

Graph Neural Networks are one of the relatively recent additions to the family of deep neural network architectures. The applications include but are not limited to physics, chemistry, biology, and linguistics [27]. GNN is now an umbrella term that incorporates a variety of different neural networks that are able to take graphs as their input and usually operate through so-called message-passing [38]. The message-passing procedure in graph neural networks is a way to aggregate graph information from some neighborhood of a graph node to be incorporated in the representation of this node in the next layer of the network. Each layer t represents an aggregation step. Mathematically, this can be expressed as (see [14]):

$$\boldsymbol{m}_v^{t+1} = \sum_{w \in N(v)} \boldsymbol{M}_t (\boldsymbol{h}_v^t, \boldsymbol{h}_w^t, e_{v,w}), \tag{8}$$

$$\boldsymbol{h}_v^{t+1} = \boldsymbol{U}_t (\boldsymbol{h}_v^t, \boldsymbol{m}_v^{t+1}), \tag{9}$$

where $\boldsymbol{h}_v^t$ refers to the vector representation of the node v at hidden layer $\boldsymbol{h}^t$, $e_{v,w}$ is a (weighted) edge present between nodes v and w, $N(v)$ is some neighborhood of the node v and $\boldsymbol{M}_t$, $\boldsymbol{U}_t$ are weight matrices that are usually to be learned in the process of network training. This message-passing scheme is applied to a graph several times to produce a representation of the graph at the last layer that is informed about the underlying graph structure in a manner that is potentially useful to the learning task at hand. Generally,

Eq. (8) represents an aggregation of the neighborhood information step and Eq. (9) updates nodes representations at the next layer by incorporating aggregated information into the nodes representations.

The **Graph Convolutional Network** is probably one of the most popular GNN architectures. It was introduced in 2017 by Thomas Kipf and Max Welling [21]. As its name suggests, GCN performs a convolution/aggregation of node information on some neighborhood. Given the graph $\boldsymbol{G}$ with the set of nodes $\boldsymbol{V}$, the set of edges $\boldsymbol{E}$ and adjacency matrix $\boldsymbol{A}$, the GCN layer can be expressed as follows:

$$\boldsymbol{H}^{t+1} = \sigma(\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{H}^{t}\boldsymbol{W}^{t}), \tag{10}$$

where $\tilde{\boldsymbol{D}}$ is a diagonal matrix with $\tilde{\boldsymbol{D}}_{ii} = \sum_j \tilde{\boldsymbol{A}}_{ij}$. $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$, where $\boldsymbol{I}$ is the $|\boldsymbol{V}| \times |\boldsymbol{V}|$ identity matrix that adds self-connections to the adjacency matrix. The activation function of choice is $\sigma(\cdot)$. $\boldsymbol{H}$ is an $|\boldsymbol{V}| \times m$ matrix, with m being the length of an embedding vector of vertex. Finally, W is a learnable weight matrix.

This operation as shown in Eq. (10) is applied several times. However, GNNs are usually shallow networks, since after performing too many convolution steps, neighborhood information propagates too far through the graph and all nodes become homogeneous. This problem is well-studied and referred to as oversmoothing [26].

GraphSAGE is a variation of a message-passing architecture on graphs and was proposed by Hamilton, Ying, and Leskovec [17]. The idea behind this architecture is very similar to the GCN layer described above, with three major differences. Firstly, the neighborhood at each training step is subsampled. Next, while GCN aggregates information over the neighborhood by taking the weighted average, GraphSAGE does not impose a particular choice of the aggregation function. The authors of the GraphSAGE architecture investigated in their paper a simple mean, an LSTM (Long Short Term Memory) layer as an aggregation function and different versions of the pooling functions applied per feature in embedding, e.g., mean and max pooling. Finally, contrary to GCN module, GraphSAGE does not incorporate information about a node's neighborhood into its representation at the next layer by aggregation, but rather by concatenation of these two different pieces of information. By doing this, more information is preserved, namely information about a node itself and its locality in the graph is kept separately [1].

2.1.5. Optimizers

The most common technique to train neural networks is by optimizing their weights with gradient descent. The plain or vanilla gradient descent suffers from several drawbacks, e.g., slow or no convergence, getting stuck in local minima and saddle points, etc. (for more details see [35]). All these shortcomings of the plain gradient descent spurred the development of various gradient descent optimization algorithms that aim to speed up the process of convergence.

Adam is one of the most popular optimizers for training neural networks, and is usually appreciated for its speed, ability to deal with sparse gradients and computational efficiency [20]. Adam accumulates the previous gradients as their moving average, keeps track of the moving average of the previously squared gradients and combines these two pieces of information to determine the size of the step at the current iteration of gradient descent.

Rprop or resilient backpropagation method was introduced by Riedmiller and Braun [34] but is rarely used in recent research. Contrary to many other step optimization algorithms, Rprop uses only the sign of the derivative and does so independently per parameter. In this research, Rprop was chosen for considerable solution time speedup that it offered.

2.2. Previous research

Many authors have recently applied GNNs to CO problems in an unsupervised manner. Karalias and Loucas [18] described a learning framework where, first, the GNN is used to fit the probability distribution over the graph nodes, which describes a probability of a node belonging to the optimal solution. In the next step, an optimal solution is recovered either by sampling or by the method of conditional expectation. The authors applied this framework to solve the maximum clique problem and a constrained min-cut problem and report that this approach outperforms many existing deep learning algorithms and classical solvers in terms of the quality of the solution and scalability. However they do not provide any information about sensitivity of the results to the density of the graphs at hand.

Yao, Bandeira and Villar [42] applied a GNN to the MaxCut problem, which can be formulated as a binary quadratic unconstrained optimization problem. They evaluated their approach on random regular graphs with low connectivity. The authors report that GNNs perform on par with other existing methods.

Xu, Hui and Fu [41] apply a variation of the GNN architecture named TilinGNN to the 2D tiling problem. The decision variable $x_i \in [0, 1]$ that is being output by the GNN is defined as the probability of a tile being placed at location i. The authors show that their approach works for several tiling sets with a linear scaling of solution time to the number of candidate placements for tiles, however they also report the inability to work with specific types of inputs such as Penrose tile sets, and inability to impose hard constraints.

Schuetz et al. [37] applied GNN to the graph coloring problem. The authors proposed and compared two GNN architectures: GCN and GraphSAGE and applied them to the graphs from two benchmark graph datasets: the COLOR dataset and the citation network datasets. Reported results for the example graphs from both datasets indicate that the combination of increasing graph size and graph density poses challenges to the GNN-solvers, however the GraphSAGE architecture seems to be more robust against these challenges. Notably, according to the authors, the superior performance of the GraphSAGE-based architecture, comes at a price of considerable training times.

In another work by Schuetz et al. [36], they applied a GCN to several CO problems formulated as QUBOs in an unsupervised manner, using the QUBO cost function as a loss to be minimized during the "training" process. In the research, this unsupervised

approach was applied to the MaxCut and the MIS problems defined on a series of generated graphs. The authors compared their solutions for the MaxCut problem to the solutions derived with the Goemans-Williamson approximate algorithm and analytically derived upper bounds. The reported cut sizes found by the GCN-solver were close to the theoretical upper bound. The GCN-solver performed similarly well on the MIS instances. However, all the experiments were performed on the graphs with degrees 3 and 5, which can be considered very sparse especially as the number of nodes grows.

The above mentioned work was subjected to the broad criticism, among others by Boettcher [6] and Angelini and Ricci-Tersenghi [3]. Both point out the lack of comparison to the trivial, baseline models such as gradient descent and greedy algorithm for MIS. Angelini and Ricci-Tersengh [3] further illustrate that MIS problem for the sparse graphs is not a representative example of the hardness of the MIS problem in general, pointing out that as Barbier et al. [5] demonstrated, MIS becomes difficult to approximate for the regular graphs with $d > 16$. From the results, presented in [36] it remains unclear if the GNN solver applied to the graphs with higher connectivity is able to produce solutions of the similar quality.

To conclude, while in recent years many specific applications of GNNs to CO problems have been studied, we find that not enough attention has been paid to the known challenges that density and symmetries of problem graphs pose. Although previously derived results indicate that these obstacles influence negatively the quality of the solution derived, their effect has not been specifically measured and no specific remedies to mitigate their influence have been suggested. With this research we aim to address these shortcomings.

3. Experiments

The architecture as proposed in literature suffers from a number of drawbacks, such as relatively long solution times and difficulty in dealing with denser graphs for the MIS problem. For this, in this section a new approach is proposed and the instances that were used to test this approach are explained.

3.1. Methods

3.1.1. Graph neural network solver

The GNN solver solution process is an unsupervised training procedure, where the QUBO formulation's cost function is used as the loss function to be minimized during the training phase. In order to effectively perform gradient descent on a QUBO cost-function, the cost-function should be continuous and twice differentiable. To meet the continuity condition we followed the usual approach from previous research [36,37,42], and replaced the binary variable $\mathbf{x} \in \{0,1\}^n$ with continuous $\mathbf{p} \in (0,1)^n$, which can be interpreted as the probability of binary variable taking value 1.

The basic version of the GNN solver follows the architecture described in [36] and is schematically represented in Fig. 1. There are 2 GCN layers implemented by PyTorch (GraphConv layers). Each node in the graph (variable) is represented by a trainable embedding vector of size $m = \sqrt{|\mathbf{V}|}$. Then the input to the solver is represented by $h_0^{(n,m)}$ and the output of the second GCN layer is $h_2 \in \mathbf{R}^n$. The scaling to the probability vector $\mathbf{p}$ is done by applying sigmoid function:

$$\mathbf{p} = \sigma(\mathbf{h}_2). \tag{11}$$

The final integer solution is obtained by a rounding function. The training is performed with the Adam optimizer.

As discussed in Section 4, this architecture suffers from a number of drawbacks, such as relatively long solution time and difficulty in dealing with denser graphs for the MIS problem. Below, we propose three adjustments to the basic architecture of the GNN solver. The results of these experiments with respect to MC and MIS problems are discussed in the next section.

1. **Change of optimizer from Adam to Rprop** In the preliminary series of experiments with Gradient Descent it was discovered that Adam is not the best optimizer across all graphs for solving the MIS problem (this point is further discussed in the Section 4). Therefore, the Adam optimizer was replaced by Rprop.
2. **Replacing GCN graph convolution module with GraphSAGE** Following [6,37] we replaced GCN module with GraphSAGE and compared the performance of the two GNN solvers on MC and MIS problems.

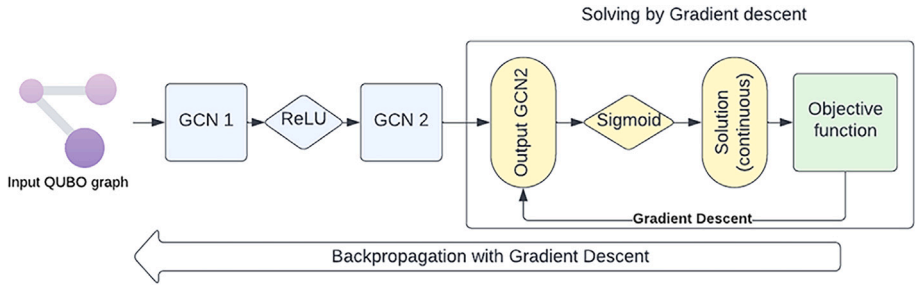

Fig. 1. Diagram of the GNN solver architecture.

3. **Transferring embeddings trained on the Max Cut problem** to the MIS problem As was discussed in Section 2.1.4, the output of a graph neural network produces a low dimensional representation for each node in the graph. In this research, this representation is given by vector **p** (see Eq. 11). However, since the solution process is performed by backpropagation through the GNN layers back to the inputs, it is also possible to define and learn graph node embeddings for a particular graph with the idea that such embeddings learned on one problem can later be used to facilitate the learning/solution process on a different but related problem defined on the same graph.

   As discussed further the GNN-solver is able to consistently find high-quality solutions for the MC problem, producing not only a solution but also potentially useful node embeddings. However, this was not the case for the MIS problem. We investigated whether using the node embeddings learned for the MC problem can facilitate the solution process for the MIS problem on the same graph.

3.1.2. Other methods

To obtain an unbiased impression of the GNN-solver's performance with respect to problem type and growing graph density, it was compared to the solutions found by known classical algorithms that are used to solve MaxCut and MIS. Both MaxCut and MIS problems are NP-hard, thus there are no known polynomial time algorithms. For MIS, even an efficient approximation algorithm does not exist. Based on the previous research described above and constructive criticisms published earlier, it was not expected that the GNN-solver would have been able to beat well known previously developed algorithms. Thus, the baseline methods were chosen not from SOTA algorithms, but from ones that were able to easily provide a solution that could be used as a reference through the series of the experiments performed. For these reasons, two heuristics were chosen, proposed in the research and implemented in the Networkx library as off-the-shelf solutions.

Heuristic for the maximum cut problem. The choice was made to utilize a simple one-exchange heuristic as implemented and provided by the NetworkX library [10], which tries to locally improve some randomly chosen cut by one-exchange strategy. At each step the algorithm greedily aims to add or swap randomly chosen nodes if this improves the size of the cut. If no improvement can be made, the algorithm terminates.

Heuristic for the maximum independent set problem. As was mentioned above, the MIS is a strongly NP-hard problem with no known efficient approximation algorithms.

In this research, a modified version of the minimum degree greedy algorithm was used as described in [7] and implemented by the Networkx library. The simple minimum degree greedy algorithm [16] was shown to have $\frac{\delta+2}{3}$ performance ratio, where δ is a maximal degree of the graph **G**. The idea of the algorithm is very straightforward: starting from the empty independent set, consequently add one by one nodes from the list of the still available nodes sorted by their degree from lowest to highest. After the node is inserted into the independent set, it and all its neighboring nodes are removed from the list of still-available nodes. The modified Networkx implementation of the algorithm replaces sorting by recursive calls on the neighborhood of a pivot node (the node chosen randomly to be included in the independent set) and its non-neighborhood and choosing the one that attains larger independent set sizes.

Gradient descent. The main tool to train neural networks in general and GNNs specifically is gradient descent—a well-known method of mathematical optimization. Since previous research, as mentioned above, points out the lack of useful gradient information as one of the challenges of applying machine learning techniques to CO problems, it is interesting to investigate whether chosen example problems suffer from this impediment and how much this hinders the GNN solver process. Specifically, there is a difference between the effects that the growing graph density has on the quality of the solution and GNN solver performance on MaxCut and MIS problems. We investigate whether this difference can be attributed to the insufficient gradient information of a particular problem by applying gradient descent to solve specified problem instances.

The last part of the solution process shown in Fig. 1 represents the gradient descent procedure used. The initial solution is selected randomly, subjected to the sigmoid function that maps it to a probability vector and then the cost of this solution is calculated. The initial solution is then modified iteratively by the gradient descent algorithm.

3.2. Problem instances

To investigate the effect of the growing density of the graph on the quality of the solution derived by the GNN solver, a collection of graphs of two types was randomly generated with the Networkx library [10]: *d-regular graphs* and *probabilistic graphs*. A *d-regular graph* is a graph in which each node has exactly d adjacent edges. To generate a *probabilistic graph*, a random Gilbert graph model was chosen in which each possible edge appears with probability p, such that $0 < p < 1$.

The graphs of different sizes and densities were studied, in particular for each type of graph a set of graphs $\mathbf{G}^{reg}(n, d)$ (regular) or $\mathbf{G}^{prob}(n, p)$ (probabilistic) was generated, where:

$$n \in \{100, 200, 300, 500, 700, 1000\}$$

and

$$d \in \{3, 4, 5, 6, 7, 9, 10, 15, 17, 18, 19, 20, 21, 22, 23, 24, 25, 30, 35, 40, 45, 50, 60, 70\}.$$

To keep the comparison between two types of graphs consistent, the parameter p was defined such that the total number of edges in $\mathbf{G}_i^{reg}(n_j, d_k)$ and expected number of edges of $\mathbf{G}_i^{prob}(n_j, p_k)$ would be the same. Here, i refers to the i-th graph in the collection, $j \in 1, 2, \ldots, 6$ is the j-th size and $k \in 1, 2, \ldots, 24$ is the k-th density.

Hardware specifications. All experiments were run on an Intel Core i7-1255U CPU @ 1.70 GHz (16 GB RAM) machine.

4. Results

The experiments with the basic version of the GNN solver revealed that its performance differs per problem type (MaxCut or MIS) and for the MIS problem is impeded by the graph symmetry and growing graph density. Fig. 2 (left) shows which fraction of the size of the MaxCut solution found by the heuristics described in 3.1.2 the GNN solver was able to find. The leftmost points correspond to the sparsest graphs and it is easy to see, that for those graphs solutions found by GNN solver even exceed benchmark solutions in cardinality. In general for both types of graphs–regular and probabilistic–it was able to find solutions to MaxCut problem of a similar quality as classical heuristics. Only on the dense graphs ($n = 100$, $d > 40$) performance is worse. However, when solving MIS the basic version of the GNN solver was able to produce solutions for very sparse graphs only. Fig. 3 shows the size of the independent set found for the collection of regular graphs. The non-empty independent set was found only for the small subset of graphs with a low degree. Fig. 2 (right) shows that as the density of the graph grows, the quality of the solution found, measured as a fraction of the solution found by a classical heuristics, deteriorates very quickly, and for graphs with $d > 7$, the solver always returned an empty set as a solution for both types of graphs irrespective of their size. The symmetries in the regular sparser graphs seem to slightly worsen the performance further compared to probabilistic graphs.

The best performance was demonstrated on the very sparse graphs, where for the both MaxCut and MIS problems the basic GNN solver, on average found larger sizes of the cut than the classical heuristics. Also, similar to the results presented in [36] solution runtimes were shorter both for MaxCut and MIS problem instances with the most noticeable improvement for the larger non-sparse MaxCut instances (see Fig. 4).

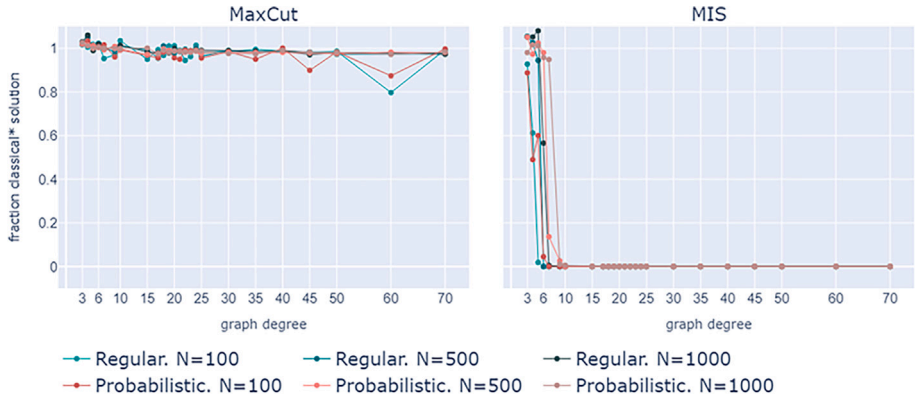

Fig. 2. Average fraction of classical* solution found by the basic GNN solver. (* "classical" refers to the solution found by the heuristics used as described in Section 3.1.2).

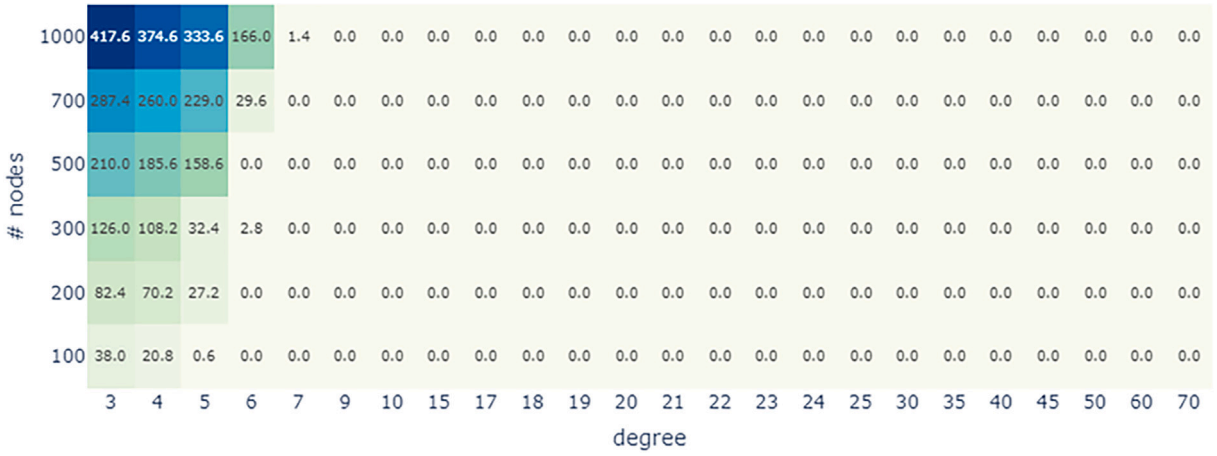

Fig. 3. Average size of the independent set found by the basic GNN solver on regular graphs.

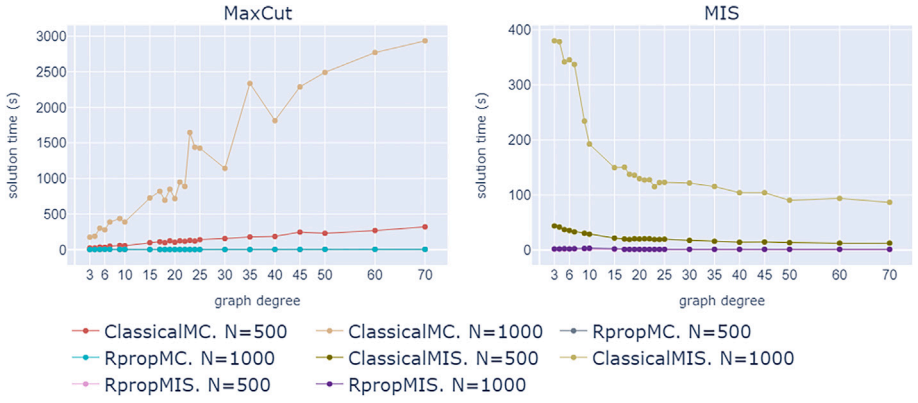

Fig. 4. Average solution time for MaxCut and MIS for classical heuristic and GNN solver with Rprop.

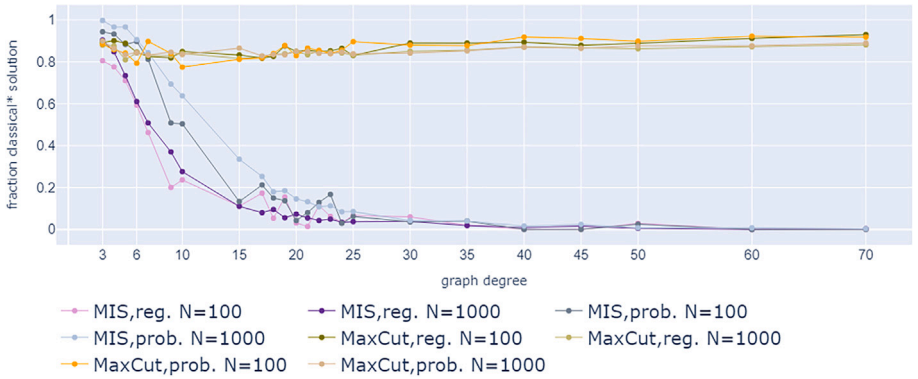

Fig. 5. Average fraction of classical* solution for MIS and MaxCut found by gradient descent. Regular and probabilistic graphs.

To shed light on the difference in performance per problem and evaluate the sufficiency of the gradient information, both problems were also solved using the gradient descent method.

Fig. 5 shows a striking difference in performance of the gradient descent approach on the MC and MIS problems. For the MC problem instances the plain gradient descent method consistently finds cuts of $80\% - -90$ % size of the cut found by the classical heuristic, while performance deteriorates very quickly for the MIS problem as the density of the graphs increases. For the graphs with a density of 30 and higher, gradient descent almost always converges to the trivial solution (empty set). The quality of the solution for the MaxCut problem found by gradient descent is less dependent on the type of the graph than on its size and respective density. However, this is not true for the MIS problem: when solving MIS problems on probabilistic graphs of comparable density, gradient descent finds slightly better solutions than on regular graphs. Especially on sparse graphs gradient descent was able to find independent sets larger than the ones found by a classical heuristic.

These results suggest that gradient information derived from the MIS QUBO formulation can guide solution process for the sparse graphs, but as the density of the graph grows, usefulness of the gradients for the solution path diminishes. These empirical results agree with the findings of [5].

Below we describe the results of the three proposed improvements to the approach.

GNN solver with Rprop optimizer Noticeably, plain gradient descent was able to deliver non-empty solutions for the MIS problem for denser graphs than basic GNN solver, which suggested that basic GNN solver was disregarding potentially useful gradient information to the detriment of the obtained solution, potentially due to the choice of the optimizer (Adam). GNN solver with Rprop optimizer performed better on MIS instances (Fig. 6, left) compared to the basic GNN solver (Fig. 2) in terms of degree of the graphs for which it was able to find non-empty independent set and showed improved solution times for both MaxCut and MIS problem instances (Fig. 6, right). The GCN-solver with Rprop optimizer was able to find good MIS solutions for graphs with $d \leq 10$ within reduced solution time. Also performance improvement for probabilistic graphs compared to the regular graphs of the same density was even more pronounced, which means that GNN solver with Rprop can better utilize asymmetry of the probabilistic graph to guide the solution process.

GraphSAGE-based GNN solver with Rprop optimizer Fig. 7 shows the resulting performance of the GNN solver with Rprop optimizer after replacement of the GCN module by GraphSAGE module. The GraphSAGE module showed better ability to tackle denser graphs for the MIS instances. In fact only on smaller graphs ($N = 100$) with higher degrees did some runs still deliver empty sets. Solving larger MIS instances with graph degrees up to 70 always resulted in non-empty independent sets. At the same time, the

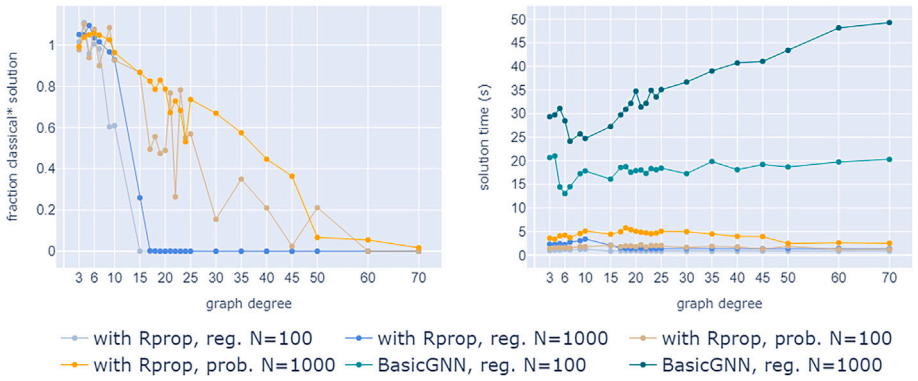

Fig. 6. GNN solver with Rprop optimizer. Left: Average fraction of classical* solution for MIS found by GNN solver with Rprop. Right: Average solution times for MIS.

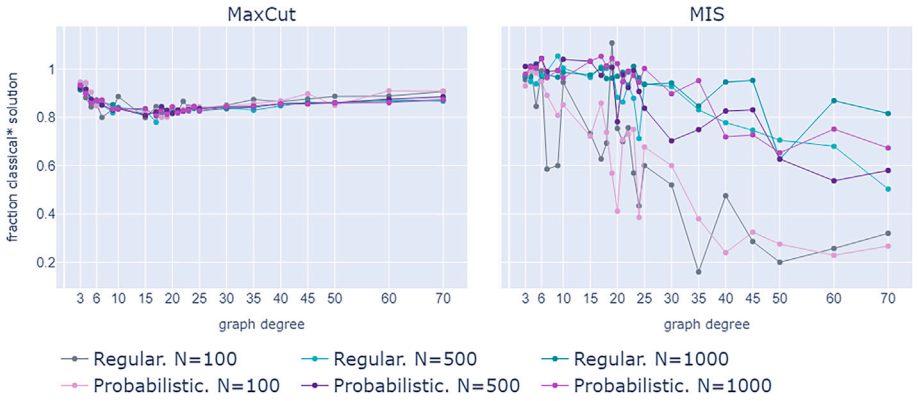

Fig. 7. Average fraction of classical* solution for MIS and MaxCut found by GraphSAGE–GNN solver.

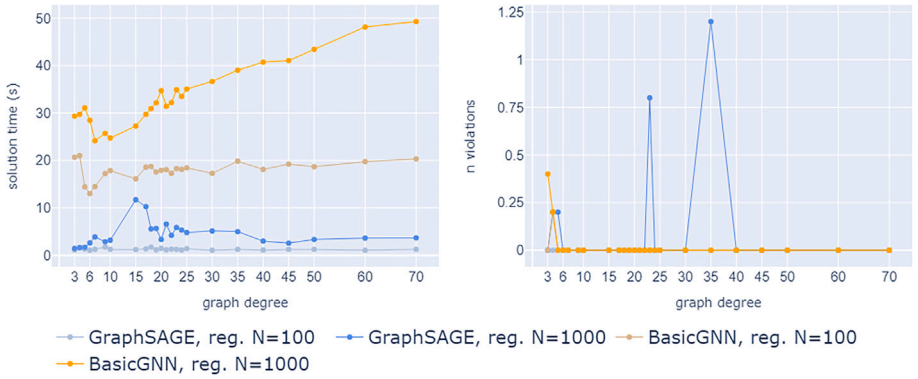

Fig. 8. GraphSAGE vs. basic GNN solver. Left: Mean solution time MIS on regular graphs. Right: Mean number of violations per MIS instance on regular graps.

average size of the cut found by the GraphSAGE-solver for MaxCut instances was smaller than the one found by the basic GNN solver. The average solution times of the GraphSAGE-solver slightly increased compared to GCN-Rprop version, but are still consistently lower than those of the basic GNN solver with Adam (see Fig. 8, left). Finally it should be mentioned that all tested GNN solver modifications produced solutions with small number of violations of the independence constraint. Solutions found by GraphSAGE for dense graphs that were not solvable by other modifications also occasionally contained a small number of violations (see Fig. 8, right), mainly for the larger graphs. The correctness of those solutions usually can easily be restored by removing one of the connected vertices in the returned set.

Despite the fact that the GraphSAGE solver found good MIS solutions on larger graphs, the solutions found on denser smaller graphs: $n = 100, d > 20$ are worse than the ones found by the used heuristics in terms of the size of the independent set found. At the same time the quality of the solutions on the larger graphs still slowly deteriorates with the growing density, which indicates that GraphSAGE-solver also cannot completely overcome the problem of finding high quality solutions on dense graphs.

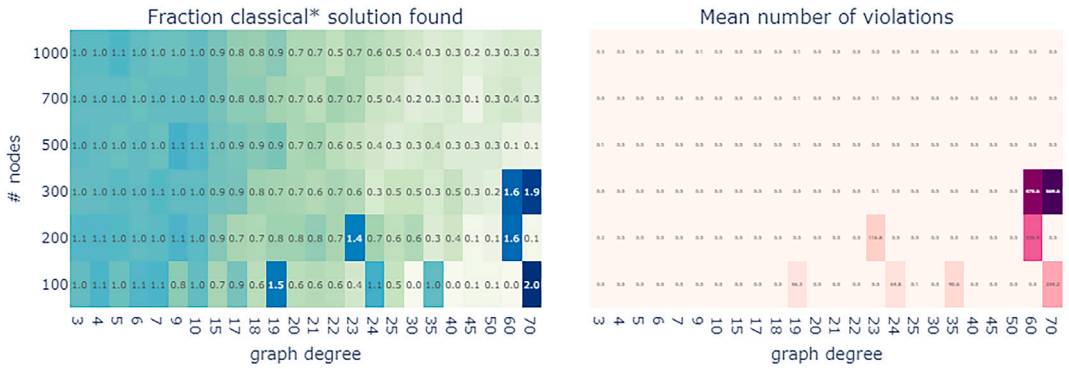

Fig. 9. GNN-solver with embedding transfer. Regular graphs. Left: Average fraction of classical* solution for MIS found. Right: Average number of violations of independence constraint per solution.

Embedding transfer Results of the gradient descent experiments indicated that gradient information on denser MIS graph instances is not sufficient to guide the solution process. The fact that basic GNN solver demonstrates similar struggles with denser graphs suggests that due to the lack of the useful gradient information, a meaningful node representation also cannot be learned during the MIS solution process. On the other hand, MaxCut seems to be usually susceptible to gradient descent and ability of the GNN solver to find good solutions for MaxCut problem instances should result in producing also useful node representations in the form of the embeddings that are being trained during the solution process. Although the MaxCut and MIS problems are not directly related, it is to be expected that the vertex subsets $\mathbf{S}_1$ and $\mathbf{S}_2$ returned as a solution to the MaxCut would have fewer inner connections, and would be more independent than just randomly sampled. Thus, node embeddings learned on the MaxCut problem might also guide a solution process for the MIS problem on the same graph.

The results of this experiment tentatively suggest that node embeddings which were trained on the MaxCut instances might indeed be useful to some degree for the MIS problem.

The range of graphs in terms of their degree for which a quality solution can be obtained was broadened (see Fig. 9, left). The overall fraction of total runs in which an empty set was returned as a solution to MIS dropped to zero for most of the graphs. However, as Fig. 9, right subplot shows, the correctness of the returned solution is not guaranteed, especially on the dense graphs where many violations of the independence constraint were detected.

Also transfer learning inevitably leads to the increased solution times since the solution time required for solving one particular instance of an MIS problem includes also a solution time for the MaxCut problem on the same graph instance to pretrain embeddings for the transfer.

When performing an embedding transfer, as described in Section 3.1.1, one has an option to freeze them, which will exclude them from the training process to solve the MIS. Frozen embeddings are treated like constant input and are not adjusted in the training loop. Another option is to unfreeze them and continue training for a problem at hand, which allows a GNN solver to adjust them according to the current loss function. Transferring frozen embeddings resulted in larger sizes of the sets returned as solutions to MIS. However, the higher cardinality of sets returned as solutions is caused by a considerable number of violations of the independence constraint. Thus, solving the MIS with frozen embeddings initially trained on the MaxCut instance consistently has a higher chance of obtaining an incorrect solution, which is to be expected, since both MaxCut and MIS problems are not one-to-one related. When embeddings are not frozen after transfer, they are able to guide the search process while being simultaneously adjusted to a different cost function.

Still, as Fig. 9 suggests the embedding transfer does not deliver high quality solutions for dense graphs, producing either smaller sets or sets with a lot of violations of the independence constraint. This result is most likely due to the fact that as the density of a graph grows, subsets $\boldsymbol{S}$ and $\boldsymbol{S}$' returned as a solution to the MaxCut problem, contain a progressively higher number of inner edges and these subsets are less independent than the ones found in the sparse graphs. Thus as density of a graph grows, the usefulness of the embeddings trained on the MaxCut problem for the MIS problem quickly diminishes. To summarize the above, although the embedding transfer might provide some useful information to the GNN-solver in some cases, this approach combined with the GCN architecture still cannot completely overcome the density hurdle. Interestingly, the GraphSAGE architecture did not benefit further from the embedding transfer, which might suggest that it is efficient enough in learning useful node representations during the MIS solution time.

5. Conclusions

In this work we investigated to which extend Graph Neural Networks can be leveraged to solve Max Cut and MIS problems formulated as QUBOs. Specifically, we looked at how the growing graph connectivity and graph symmetry influence the GNN-solver performance per problem type and to what extent the discovered negative effects of graph density on the quality of the solution to the MIS problem can be mitigated by different GNN architectures and transfer learning.

It was found that the basic GNN-solver architecture proposed by Schuetz et al. [36] when applied to the MIS problem can only handle very sparse graphs. As our experiments with gradient descent suggest, and as was previously pointed out in multiple studies,

the gradient information obtained on discrete problems is not always useful and cannot always guide a solution process. However, this seems to depend on the problem at hand. The Max Cut problem instances were susceptible to gradient descent and thus solvable by the GNN-solver irrespective of the density of the underlying graph: GNN-solver was able to find good quality solutions and within reasonable solution time budget even for larger graph instances. However, the MIS problem instances were not. The increasing degree of the graph for the initially constrained MIS problem, especially combined with its regularity, seems to pose an obstacle for a GCN-based architecture.

To mitigate these negative effects we proposed and tested several adjustments to the basic architecture and measured their robustness against growing density of the MIS QUBO graph. Experimental results indicate that these adjustments can generally broaden the scope of MIS graph instances that can be efficiently solved in terms of their increasing density; however, none of these modifications were able to ultimately overcome the growing graph density hurdle.

Switching from Adam to Rprop optimizer allowed on the one hand solving denser graph instances and on the other considerably reduced solution times. Given the initialization dependent nature of the GNN-solver, considerable reduction in solution time offers possibility of more solution runs and thus broader search through the solution space, which positively contributes to the quality of the best found solution.

GraphSAGE layers: Replacing GCN layers with GraphSAGE layers showed the best results in broadening the scope of the MIS instances that can be solved in terms of increasing graph degree. However, GraphSAGE-based solvers are still impeded by the growing degree of the graph and failed to find a solution of comparable quality to the ones found by classical heuristics for the high-degree graphs. This can be due to the general limitations of the GNN architectures, as pointed out in previous research [8,13,31].

Embedding transfer: Learning graph representations on the MaxCut problem and transferring them to the MIS instances enhanced the ability of the GCN solver to solve denser graph problems. However, usefulness of the embedding diminishes with the growing density of the graph introducing increasing number of violations of the independence constraint into the solution.

Among the general criticisms that the GNN solver proposed in [36] was subjected to, Gamarnik [12] states that as a local algorithm GNN is unlikely to demonstrate good performance on the hard instances. While the results presented in this paper are consistent with that statement, we point out that as shown above addition of the global information about the solution space through transfer learning and specifically identification of the potentially relevant problems defined on the same graphs that can be used to train informative graph node embeddings might be a promising direction for the future research.

In conclusion, this study makes the following key contributions:

Systematic evaluation of GNN solver on QUBO-formulated CO problems –We investigate in detail the performance of GNN solver for the MaxCut and MIS problems across a wide range of graph sizes, densities and symmetries, identifying problem-specific challenges.

Identification of density- and symmetry-related performance limits –We quantify how graph connectivity and regularity affect solution quality, especially for MIS problem instances, and link these effects to limitations in gradient information.

Proposed architectural and training improvements –We introduce and experimentally assess three techniques to improve GNN solver robustness for MIS instances on dense graphs: (i) replacing Adam with Rprop optimizer, (ii) replacing GNN layers with GraphSAGE layers, and (iii) applying transfer learning between MaxCut and MIS instances.

Comprehensive comparative analysis –We benchmark the proposed approaches against classical heuristics and gradient descent baselines, providing insights into when and why GNN solver succeeds or fails.

Together, these contributions provide both a deeper understanding of the limitations of unsupervised GNN solver for QUBO-based CO problems and practical strategies to broaden their applicability.

CRediT authorship contribution statement

Olga Krylova: Writing – original draft, Visualization, Methodology, Conceptualization. **Frank Phillipson:** Conceptualization, Supervision, Writing - review & editing

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.

Acknowledgement

This work was supported by the Dutch National Growth Fund (NGF), as part of the Quantum Delta NL programme.

References

[1] C.C. Aggarwal, et al., Neural networks and deep learning, Springer 10 (2018) 3.
[2] B. Amos, J.Z. Kolter, Optnet: differentiable optimization as a layer in neural networks, in: International Conference on Machine Learning, PMLR, 2017, pp. 136–145.

[3] M.C. Angelini, F. Ricci-Tersenghi, Modern graph neural networks do worse than classical greedy algorithms in solving combinatorial optimization problems like maximum independent set., Nat. Mach. Intell. 5 (2023) 29–31.
[4] M. Anthony, E. Boros, Y. Crama, A. Gruber, Quadratic reformulations of nonlinear binary optimization problems, Math. Program. 162 (2017) 115–144.
[5] J. Barbier, F. Krzakala, L. Zdeborová, P. Zhang, The hard-core model on random graphs revisited, in: Journal of Physics: Conference Series, IOP Publishing, 2013. p. 012021.
[6] S. Boettcher, Inability of a graph neural network heuristic to outperform greedy algorithms in solving combinatorial optimization problems, Nat. Mach. Intell. 5 (2023) 24–25.
[7] R. Boppana, M.M. Halldórsson, Approximating maximum independent sets by excluding subgraphs, BIT Numer. Math. 32 (1992) 180–196.
[8] Q. Cappart, D. Chételat, E.B. Khalil, A. Lodi, C. Morris, P. Veličković, Combinatorial optimization and reasoning with graph neural networks, J. Mach. Learn. Res. 24 (2023) 1–61.
[9] K. Chandrasekaran, R. Karp, E. Moreno-Centeno, S. Vempala, Algorithms for implicit hitting set problems, in: Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2011, pp. 614–629.
[10] N. Developers, NetworkX. Approximations and heuristics, https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.approximation.maxcut.randomized_partitioning.html#networkx.algorithms.approximation.maxcut.randomized_partitioning
[11] P. Donti, B. Amos, J.Z. Kolter, Task-based end-to-end model learning in stochastic optimization, Adv. Neural Inf. Process. Syst. 30 (2017).
[12] D. Gamarnik, Barriers for the performance of graph neural networks (GNN) in discrete random structures, Proc. Natl. Acad. Sci. 120 (46) (2023) e2314092120.
[13] V. Garg, S. Jegelka, T. Jaakkola, Generalization and representational limits of graph neural networks, in: International Conference on Machine Learning, PMLR, 2020, pp. 3419–3430.
[14] J. Gilmer, S.S. Schoenholz, P.F. Riley, O. Vinyals, G.E. Dahl, Message passing neural networks, Machine Learning Meets Quantum Physics (2020) 199–214.
[15] F. Glover, G. Kochenberger, Y. Du, A tutorial on formulating and using qubo models. arXiv preprint arXiv:1811.11538, 2018.
[16] M. Halldórsson, J. Radhakrishnan, Greed is good: approximating independent sets in sparse and bounded-degree graphs, in: Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 1994, pp. 439–448.
[17] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, Adv. Neural Inf. Process. Syst. 30 (2017).
[18] N. Karalias, A. Loukas, Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs, Adv. Neural Inf. Process. Syst. 33 (2020) 6659–6672.
[19] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, L. Song, Learning combinatorial optimization algorithms over graphs, Adv. Neural Inf. Process. Syst. 30 (2017).
[20] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.
[21] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907, 2016.
[22] G.A. Kochenberger, F. Glover, A unified framework for modeling and solving combinatorial optimization problems: a tutorial, Multiscale Optim. Methods Appl. (2006) 101–124.
[23] W. Kool, H. Van Hoof, M. Welling, Attention, learn to solve routing problems! arXiv preprint arXiv:1803.08475, 2018.
[24] J. Kotary, F. Fioretto, P. Van Hentenryck, B. Wilder, End-to-end constrained optimization learning: a survey. arXiv preprint arXiv:2103.16378, 2021.
[25] H. Lemos, M. Prates, P. Avelar, L. Lamb, Graph colouring meets deep learning: effective graph neural network models for combinatorial problems, in: 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI), IEEE, 2019, pp. 879–885.
[26] Q. Li, Z. Han, X.M. Wu, Deeper insights into graph convolutional networks for semi-supervised learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2018.
[27] X. Li, Y. Cheng, Understanding the message passing in graph neural networks via power iteration clustering, Neural Networks 140 (2021) 130–135.
[28] Z. Li, Q. Chen, V. Koltun, Combinatorial optimization with graph convolutional networks and guided tree search, Adv. Neural Inf. Process. Syst. 31 (2018b).
[29] J. Mandi, T. Guns, Interior point solving for LP-based prediction+ optimisation, Adv. Neural Inf. Process. Syst. 33 (2020) 7272–7282.
[30] M.M. Mohades, M.H. Kahaei, An efficient Riemannian gradient based algorithm for max-cut problems, IEEE Trans. Circuits Syst. II Express Briefs 69 (2021) 1882–1886.
[31] C. Morris, M. Ritzert, M. Fey, W.L. Hamilton, J.E. Lenssen, G. Rattan, M. Grohe, Weisfeiler and Leman go neural: higher-order graph neural networks, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2019, pp. 4602–4609.
[32] M. Prates, P.H. Avelar, H. Lemos, L.C. Lamb, M.Y. Vardi, Learning to solve NP-complete problems: a graph neural network for decision TSP, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2019, pp. 4731–4738.
[33] A.P. Punnen, The Quadratic Unconstrained Binary Optimization Problem, Springer, 2022.
[34] M. Riedmiller, H. Braun, A direct adaptive method for faster backpropagation learning: the Rprop algorithm, in: IEEE International Conference on Neural Networks, IEEE, 1993, pp. 586–591.
[35] S. Ruder, An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747, 2016.
[36] M.J. Schuetz, J.K. Brubaker, H.G. Katzgraber, Combinatorial optimization with physics-inspired graph neural networks, Nat. Mach. Intell. 4 (2022) 367–377.
[37] M.J. Schuetz, J.K. Brubaker, Z. Zhu, H.G. Katzgraber, Graph coloring with physics-inspired graph neural networks, Phys. Rev. Res. 4 (2022) 043131.
[38] J. Tang, R. Liao, Graph Neural Networks for Node Classification, Springer Nature Singapore, Singapore, 2022, pp. 41–61.
[39] N. Vesselinova, R. Steinert, D.F. Perez-Ramirez, M. Boman, Learning combinatorial optimization on graphs: a survey with applications to networking, IEEE Access 8 (2020) 120388–120416.
[40] M. Vlastelica, A. Paulus, V. Musil, G. Martius, M. Rolínek, Differentiation of blackbox combinatorial solvers. arXiv preprint arXiv:1912.02175, 2019.
[41] H. Xu, K.H. Hui, C.W. Fu, H. Zhang, Tilingnn: learning to tile with self-supervised graph neural network. arXiv preprint arXiv:2007.02278, 2020.
[42] W. Yao, A.S. Bandeira, S. Villar, Experimental performance of graph neural networks on random instances of max-cut, in: Wavelets and Sparsity XVIII, SPIE, 2019, pp. 242–251.