Određivanje minimalnog nezavisnog dominantnog skupa primjenom grafovskih neuronskih mreža

Kozlik, Marko

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 80

MINIMAL INDEPENDENT DOMINATING SET CALCULATION BASED ON GRAPH NEURAL NETWORKS

Marko Kozlik

Zagreb, June 2024

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 80

MINIMAL INDEPENDENT DOMINATING SET CALCULATION BASED ON GRAPH NEURAL NETWORKS

Marko Kozlik

Zagreb, June 2024

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Zagreb, 04 March 2024

MASTER THESIS ASSIGNMENT No. 80

Student:	**Marko Kozlik (0036522612)**
Study:	Information and Communication Technology
Profile:	Control Systems and Robotics
Mentor:	prof. Stjepan Bogdan

Title: **Minimal independent dominating set calculation based on graph neural networks**

Description:

The goal of this thesis is to develop a method for determining the minimum independent dominant set (MIDS) in general graphs based on graph neural networks (GNN). To achieve this goal, it is first necessary to study the mathematical definition and properties of MIDS, as well as existing methods for finding it. Furthermore, it is necessary to become familiar with the basic models of graph neural networks and to make a brief overview of the field of their applications in optimization problems. In the next step, a data set should be generated, which will consist of a large number of different graphs and their MIDS, and will be used to train a custom or one of the existing GNN models. The developed method will be validated by comparing the accuracy and speed with existing methods, and its potential application to real problems such as determining the most influential nodes in multi-agent systems will be considered.

Submission date: 28 June 2024

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 80

Pristupnik:	**Marko Kozlik (0036522612)**
Studij:	Informacijska i komunikacijska tehnologija
Profil:	Automatika i robotika
Mentor:	prof. dr. sc. Stjepan Bogdan

Zadatak: **Određivanje minimalnog nezavisnog dominantnog skupa primjenom grafovskih neuronskih mreža**

Opis zadatka:

Cilj ovog diplomskog rada je razviti metodu za određivanje minimalnog nezavisnog dominantnog skupa (MIDS) u općenitim grafovima zasnovanu na grafovskim neuronskim mrežama (GNN). Za postizanje ovog cilja, potrebno je najprije proučiti matematičku definiciju i svojstva MIDS-a, kao i postojeće metode za njegovo pronalaženje. Nadalje, potrebno je upoznati se s osnovnim modelima grafovskih neuronskih mreža i napraviti kratak pregled područja njihovih primjena u optimizacijskim problemima. U sljedećem koraku treba generirati skup podataka koji će se sastojati od velikog broja različitih grafova i njihovih MIDS, a služit će za treniranje vlastitog ili nekog od postojećih modela GNN-a. Razvijena metoda validirat će se uspoređujući točnost i brzinu s postojećim metodama te će se razmotriti njena potencijalna primjena na stvarne probleme poput određivanja najutjecajnijih čvorova u više-agentskim sustavima.

Rok za predaju rada: 28. lipnja 2024.

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER'S THESIS No. 080

Minimum independent dominating set calculation based on graph neural networks

Marko Kozlik

Zagreb, August 2024.

Zahvaljujem svim profesorima, a pogotovo prof. dr. sc. Stjepanu Bogdanu na mentoriranju. Također, zahvala ide i asistentu Marku Križmančiću čiji savjeti su pomogli u izradi ovoga rada. Zahvaljujem svim prijateljima koje sam stekao tijekom studija te koji su moj studentski život činili zabavnijim. Posebna zahvala ide mojoj obitelji i djevojci Aneti koji su me podupirali cijelo ovo vrijeme.
Bez vas ovo ne bi bilo moguće.

Contents

1. Introduction

In today's world people are more and more interested in efficiency. One of the topics that is present today is optimally spreading information. That means the fastest and least costly methods for the propagation of information. This concept has applications in various disciplines such as connecting people in social networks or in multi-robot systems for spreading information about the system. In mathematical formulation, this concept belongs to graph theory problems. In connected graphs, there is a way to select nodes that can spread information to all other nodes in a graph. The optimal solution for such a graph is a minimum independent dominating set (MIDS).

MIDS is one of the problems in the field of graph theory which is an NP-hard problem. That means it can only be solved in exponential time (not in polynomial time). That is why most researchers today are trying to find new ways to determine which set in a graph could be MIDS. While it is a complex problem to find a solution, once there is a solution it is not complex to find whether it is a correct solution or not. NP-hard problems with that property are defined as NP-complete.

Recently the broader population was made familiar with neural networks by the introduction of Chat-GPT. A unique type of neural network is called Graph Neural Network (GNN) which has applications in vast areas of recent research such as physics simulations (Sanchez-Gonzalez et al., 2020), detecting fake news on social media (Monti et al., 2019) and systems for recommending a large number of items to lots of users (Eksombatchai et al., 2017). According to a paper by Pontoizeau et al. (2021) it is also possible to solve combinatorial graph problems using Graph Neural Networks (GNN). They used a GNN structure to determine the solution to the maximum independent set problems which is a problem similar to MIDS. They achieved scores above 800 and even up to 968 on a graph with 2004 nodes. This means they found a set of 968 independent nodes. The success of their results motivated us to apply GNNs to solving MIDS and explore how different parameters affect the feasibility of the solution.

In this thesis, we will first take a look at the theoretical background of MIDS and its applications in the real world by also mentioning different approaches to solving MIDS. Secondly, we will explain what GNNs are and which problems can be solved by the implementation of a GNN. Furthermore, we will provide an approach in which we attempt to solve the MIDS problem using GNN. Lastly, we will show results by comparing different structures and parameters and make some comments about further possibilities and improvements.

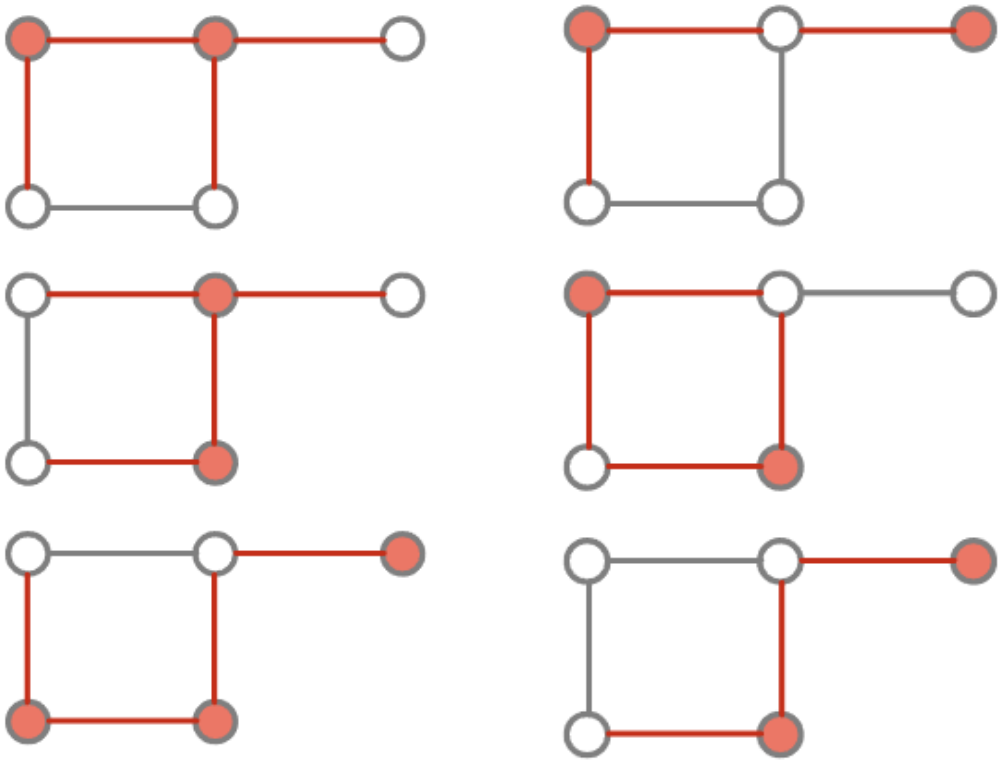
2. Minimum independent dominating set

A lot of things in today's world can be represented as graphs such as social media or maps. Graphs consist of nodes that are connected by edges. Egdes can be either directed or undirected. A directed edge has a direction from one node to another, while an undirected edge suggests a bilateral connection. If all edges of a graph are undirected then we are talking about an undirected graph. The neighborhood of a node in a graph consists of all nodes to which that node has an edge. Additionally, a graph can be weighted or unweighted. If all edges of a graph are treated the same then the graph is unweighted. In weighted graphs, edges have weights that can be comprehended as costs or the importance of each connection. A graph whose nodes are connected in a way that there is a path from each node to every other node is defined as a connected graph.

Having an undirected and connected graph G with a set of vertices V and edges E we can define a dominating set $D \subseteq V$ where for every vertex in set $V \setminus D$ there is at least one adjacent vertex from set D. Independent set I is a subset of vertices V where each pair of vertices does not contain the same edge. In other words, an independent set is the one in which no two vertices are adjacent. Combining two previous statements about dominating and independent sets we can define an independent dominating set (IDS) as a dominating set D with a constraint that it is also independent.

For every IDS in a graph, there is a set that contains the smallest number of vertices. That set is called the minimum independent dominating set (MIDS).

Figure 2.1 shows the visual representation of dominating set and independent set. By combining those properties we get an independent dominating set. All independent dominating sets of this graph are shown in Figure 2.2. It can be seen that the minimum number of nodes needed to form the IDS is 2. Because of that, it can be concluded that the first two graphs in Figure 2.2 are the solution to the



(a) Dominating set **(b)** Independent set

Figure 2.1: Dominating vs independent set

MIDS problem, while the third graph has more nodes in solution and consequently is not MIDS.

2.1. MIDS applications

Finding a solution to the MIDS problem can be useful in many areas. By identifying nodes that make the MIDS, the information can be spread to all nodes in the graph in the most efficient way. That is because every node in the graph that is not in MIDS is connected to at least one node that is in MIDS. For example, in a

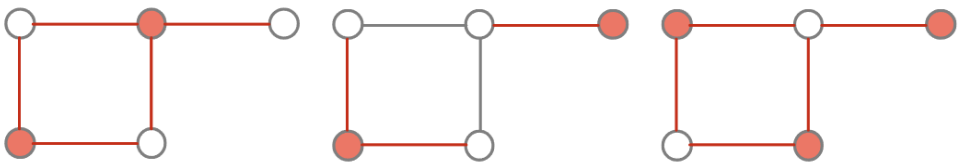


Figure 2.2: Independent dominating set

large multi-robot system, information about the object of interest can be spread by identifying MIDS. Every robot can be viewed as a node and the whole system can be represented as a graph. Edges in the graph symbolize communication between robots. The master of the whole system (a central computer) only needs to spread information to robots which make the MIDS and those robots will spread information to all other robots in a single communication step. Similar to multi-robot systems are sensor networks which consist of sensor nodes that communicate the measured values among themselves. To spread information to all nodes or retrieve sensor data at the least expense, it is useful to know the MIDS of a given network. Another and more recent application of MIDS is related to suppressing the COVID-19 epidemic. Wang et al. (2021) have introduced minimum dominating set as observer nodes to stop the epidemic spreading in the most cost-efficient way. Shen and Li (2010) have shown that MIDS can be used for multi-document summarizing as an alternative to more used approaches. Every social network can be represented as a huge graph where every node is one account whereas edges are connections between them. Having a solution to the MIDS of that network can be used for the swift and efficient spread of information.

2.2. MIDS approaches

In recent times there have been a lot of different approaches for finding the solution for MIDS in the shortest time. This problem is challenging because it is an NP-hard problem. To phrase it differently, the solution to the MIDS problem can only be found in exponential time not in polynomial time as it was noted in the article by Grandoni (2006). That means that for larger graphs, time complexity rises exponentially which can lead to massive problems when handling near real-time systems. Because of that, there have been a lot of different approaches to solving this problem in the fastest possible time. While some researchers chose the approach that includes greedy algorithms such as GRASP (Wang et al., 2017), others came up with different methods such as the local search algorithm in the k-neighborhood (Haraguchi, 2018) and the branch-and-reduce algorithm used by Gaspers and Liedloff (2012). Another alternative technique was done in a master thesis by Domislović (2022) who used a non-exact genetic algorithm to calculate MIDS. His solution worked perfectly on graphs up to 10 nodes, but with larger graphs, the accuracy was lower.

As there are a lot of combinatorial problems in graphs similar to MIDS, there

have been a lot of attempts to solve them. One similar problem is the maximum independent set. Blum et al. (2015) approached solving this problem by using bio-inspired algorithms called FrogCOL and FrogMIS. Maximum independent set was also the topic of an article by Pontoizeau et al. (2021) in which they used a graph neural network to solve this problem. They used a combination of GIN and Conv1D layers in their GNN while this thesis uses GAT and Linear layers.

3. Graph neural networks

3.1. Neural networks

Nowadays, more than ever, neural networks are being studied and utilized in a wide range of applications like image recognition for autonomous driving and pattern recognition on social media sites. Neural networks are a bio-inspired concept that was formed around the mid-20th century to formulate an artificial structure based on the structure of the human brain. The structure of a neural network includes the input layer, hidden layers, and the output layer as shown in Figure 3.1. Connections between them are trained to determine the weight of each neuron in one layer to each neuron in the following layer. The weight is a term that describes the importance of a connection and by training a neural network the weights are adjusted to give the best possible result on the output. The input

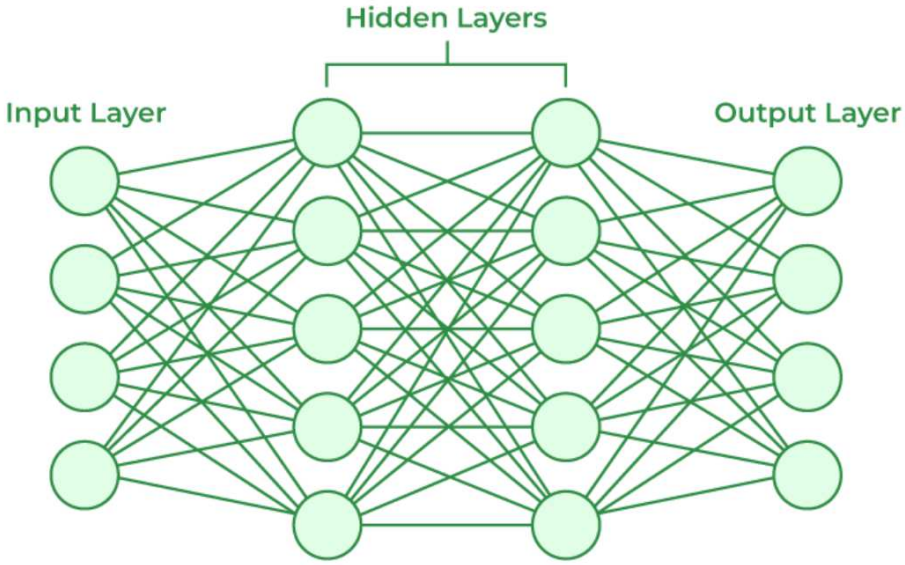


Figure 3.1: Neural Network - taken from https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications

layer contains features that are used to describe an object of interest. For example, when talking about image processing, features can be color or gradient between pixels. Every neural network has multiple features as inputs to help it to make correct predictions. The output layer of a neural network contains predictions.

When talking about supervised learning, those predictions are compared to the data's true label. By comparing prediction and labels through the loss function, the neural network trains to minimize the loss and produce more accurate predictions. To train a neural network it is needed to collect a lot of data on which the model is trained.

3.2. GNN

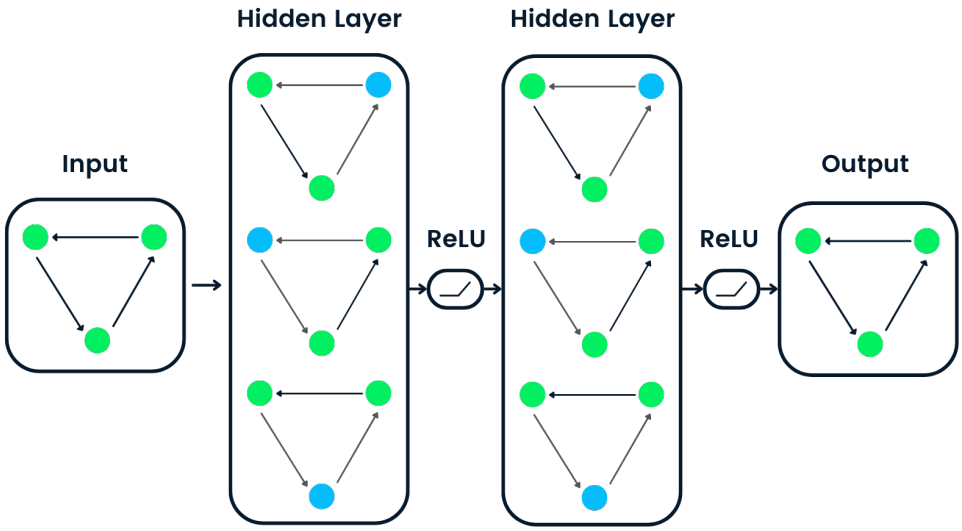


Figure 3.2: Graph Neural Network - taken from https://www.datacamp.com/tutorial/comprehensive-introduction-graph-neural-networks-gnns-tutorial

Graph neural networks are a type of neural networks which are specialized for working with graphs. They can take a graph as an input and give it as an output. The structure of a GNN is shown in Figure 3.2. A lot of points of interest can be represented as graphs. For example, a picture is a graph in which every pixel is a

node and adjacent nodes compose edges, a molecule can be described as a graph with atoms being nodes and bonds being edges.

Every graph can be defined with a square matrix that has a size of n, where n is the number of nodes in a graph. Every row and every column represents one node. That matrix is called the adjacency matrix. Each field in the adjacency matrix is either one or zero depending on whether there is an edge between those nodes. When looking at the graph in Figure 3.3, the adjacency matrix looks like this:

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}. \tag{3.1}$$

For undirected graphs, the adjacency matrix will always be symmetric and if there are no loops in a graph (if no node is connected to itself), the diagonal of an adjacency matrix will contain all zeros. Representing graphs with an adjacency matrix can be quite inefficient. As the size of the graph increases, the associated adjacency matrix size increases with a square of the number of vertices. An adjacency matrix is usually a sparse matrix which means it mostly has a lot of fields that contain zero. Because of that, there are other ways to depict a graph. One such way is called the adjacency list. It is a list that contains a list for every node that contains all nodes connected to that node. On the graph in Figure 3.3 the adjacency list looks like shown in table 3.1. This way graphs can be stored in memory more efficiently.

Table 3.1: Adjacency List corresponding to the graph in Figure 3.3

Vertex	Adjacent Vertices
1	2, 4
2	1, 3, 5
3	2
4	1, 5
5	2, 4

3 types of classification tasks can be performed on graphs: node, edge, and graph classification. When talking about node classification we are talking about the property of each individual node in the whole graph. An example of this classi-

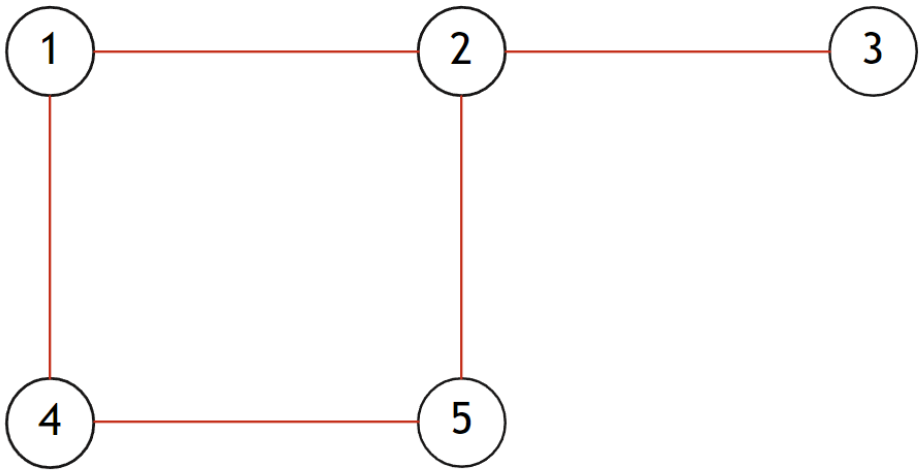


Figure 3.3: Graph with 5 nodes

fication would be categorizing different words in a document. Edge classification's task is to determine to which class each connection belongs. For example, if an edge is a part of the shortest path from one node to another. Graph classification is used for classifying a whole graph such as in chemistry for determining whether a molecule is toxic or not.

Every graph can be described with certain features. Features can be associated with nodes, edges, or the graph as a whole. For example, one node feature could be the number of adjacent nodes. Edge feature could be if an edge is directed or undirected and for a whole graph a feature can be the number of nodes in a graph. Depending on which problem needs to be solved, different features are used. Those features are passed to the first hidden layer and the feature vector is transformed into an embedding vector.

When talking about learning in graphs there has to be a way to spread information throughout the graph. GNNs are specific in a way that the information spreads between nodes not just depending on the layers of the network but also depending on the input graph. The term for that is a message passing by which every node gets information from all its adjacent nodes. The most common type of information gathering is called aggregation. Aggregation sums up embeddings of all adjacent nodes and stores them for the next iteration after passing them through an update function. That update function is different depending on which type of GNN layer is used. Other types of information gathering include average calculation or determining the maximum value.

There are many types of GNN layers. According to Daigavane et al. (2021)

these are the most popular layer types:

- GAT - Graph Attention Network (Veličković et al., 2018)
- GCN - Graph Convolutional Network (Kipf and Welling, 2017)
- GraphSAGE - Graph Sample and Aggregate (Hamilton et al., 2017)
- GIN - Graph Isomorphism Network (Xu et al., 2019)

Each of these layers has a different update function which calculates the embedding for the next step.

4. Approach

4.1. Environment

In every thesis, it is important to choose the right working environment. For training a neural network it is essential to have a high-performance graphic processing unit (GPU) as it increases the efficiency of training a model by reducing the time needed to make required computations. Training a neural network can be brought down to linear algebra because it uses matrix multiplication. GPU is most commonly used for loading and processing graphics on a computer display. Processing computer graphics is also a linear algebra problem as it uses matrices of pixels to produce a picture on the screen. Oh and Jung (2004) were one of the first who used a GPU for training a neural network and concluded its superiority against CPU (Central Processing Unit) computation. In this work, we used the Jupyter Notebook on SRCE (*cro. Sveučilišni računski centar*) server for advanced computing[1] as the working environment. Advanced computing provides computing infrastructure for solving computational resource-demanding problems. Working on their server, users have access to a powerful GPU NVIDIA A100 with 40 GB memory capable of solving demanding calculations in a faster time.

Jupyter Notebook is a platform for segmented coding with an interactive user interface. The whole process from dataset generation to training and testing was done using Python programming language. One of the most popular libraries for working with neural networks in Python is called PyTorch. This library enables users to perform tensor computations on GPU. In machine, learning tensors are a data structure similar to arrays but specialized for usage in training machine learning models as they can store multiple dimensions of data.

For testing purposes we used the Weights & Biases platform (Biewald, 2020). This platform allows developers of neural network models to perform multiple

[1] https://www.srce.unizg.hr/en/advanced-computing

tests intuitively and gives them the ability to view and customize results.

4.2. Node and graph classification

As mentioned in Section 3.2, there are three types of classifications in graph neural networks: node, edge, and graph classification. The task in this thesis is to find MIDS by using GNN. Each node can be classified either correctly or incorrectly, as a part of MIDS or not. Because of that, this is mainly a node type of classification. Globally, on the graph level, each set in a graph can either be MIDS or not. Because of that, there is also an argument to say this problem is also a (sub-)graph classification.

4.3. The challenge of multiple solutions

MIDS is a complex problem to solve in graph theory. For any graph, there is at least one, but mostly there is more than one MIDS. If we take all undirected graphs up to 8 nodes we get just over 12000 graphs of which around 3100 have a unique solution to the MIDS problem. With that in mind, the dataset can be segregated to include only those graphs. By having a dataset formulated like that and split into train and test datasets we later show that it is possible to train a GNN to have a 95% accuracy on the test set. With that in mind, it can be concluded that it is possible to train a GNN to find the MIDS on graphs. The main problem with this approach is that a model trained in this way cannot consistently find a correct solution on graphs with multiple possible MIDS solutions.

To overcome that challenge it is necessary to approach a problem differently. As it is not trivial to train a neural network with multiple correct solutions here are a few examples to undertake this challenge. One way is to expand the dataset so that for every possible solution we have a graph. This means that if for one graph there are 3 solutions to the MIDS problem, the dataset will have 3 instances of that graph but every with a different solution. Expanding the dataset in this way increases the number of graphs in the dataset from 12000 to 40000. Training a model in this fashion is a bit confusing to the neural network. When the GNN encounters the identical graph as before, the given correct solution will be different, which leads to contradiction and inability of GNN training. By approaching this problem using only one solution for every graph it is possible to train the neural network without contradiction. For each graph, a single correct solution must be

selected for training. The drawback of this approach is that while the network can be trained to find a solution to the problem it will find a specific one and will not be able to find other solutions. To overcome this, we needed to formulate a dataset differently.

4.4. Dataset generation

One of the most important steps in neural network training is to choose an adequate dataset. As the number of graphs increases exponentially with a larger number of nodes, for training the number of graphs has to be limited. In this thesis, it was chosen that the limit is around 40000 graphs. Chosen dataset contains all graphs up to 8 nodes including another 30000 graphs of size 9. That means the size of the dataset is just above 42000 or to be precise 42111 graphs.

4.4.1. Features

For every dataset, it is important to choose the right features that represent each piece of data. Features need to be chosen to represent data uniquely so that the model can be trained as adequately as possible. Features can be assigned to nodes, edges or graphs. As this is mainly a node classification all features are associated with nodes:

- degree
- degree centrality
- betweenness centrality
- disjunction value
- average neighbor degree
- closeness centrality
- 2 random features.

Each of these features describes each node in each graph in a certain way. The degree of every node is a number that represents the number of edges connected to that node. This feature is important because a node that has a higher degree is more likely to be a part of the MIDS. Degree centrality is similar to a degree but normalized in the range of 0 to 1 in a way that 1 is the value of most connections to any node in a graph. As mentioned before for a degree, it is valuable in a way

that higher values have a higher probability of formulating the MIDS. Betweenness centrality describes how central each node is when looking at the whole graph. It is a normalized value that represents how many shortest paths pass through each node. The shortest path is the one that needs a minimum number of hops between nodes to get from one node to another. Disjunction value is a more complex feature that can help a graph neural network find a correct solution. It utilizes properties of the adjacency matrix (A) and support vector. Support vector d is a vector in which i-th value is 1 if vertex i is included in the potential MIDS, otherwise 0. It can be shown that if $(\mathbf{A}+\mathbf{I})\cdot\mathbf{d}=\mathbf{1}$ the set corresponding to d is certainly MIDS. Even though the opposite does not hold, by finding a d that optimizes the given relation, we can get a heuristic for a probability that the vertex is included in MIDS. The average neighbor degree is a feature that describes the neighborhood of each node. It is a number that provides the average degree for every node in the node's immediate neighborhood. This value when combined with values like degree also shows how likely the node is to be a part of the MIDS. Closeness centrality is a feature whose value is the reciprocal value of the average shortest path from a node to every other node. It is given with a formula: $C(u)=\frac{n-1}{\sum_{v=1}^{n-1}d(v,u)}$, where n is number of nodes, u is the current node, and $d(v,u)$ is the shortest path between two nodes.

Random features were added as a random position in 2D space such that one random feature is the abscissa coordinate and the other is the ordinate coordinate. According to Sato et al. (2021), this helps in strengthening a GNN by giving every node a distinct value. The easiest way to visualize this is in a symmetrical graph such as the one shown in Figure 4.1. This graph is symmetrical and cyclical which means that every node in the graph will have the exact same feature values for any deterministic feature such as degree or centrality. Furthermore, this graph has 3 correct solutions to the MIDS problem. The solution can be any one of the nodes. When talking solely about symmetries, Figure 4.2 shows one symmetrical graph with 2 correct solutions. A solution can be either node 2 or node 3. Without having a random feature, nodes 2 and 3 would have the exact same feature values as would nodes 1 and 4. With that in mind, random features can steer the GNN in a certain direction to find a solution.

Table 4.1 shows the relative time needed for the calculation of each of the features. According to that, the most complex feature to calculate is betweenness centrality. Disjunction value takes the second most time of chosen features. This was anticipated as to calculate this value it is needed to apply matrix multiplication

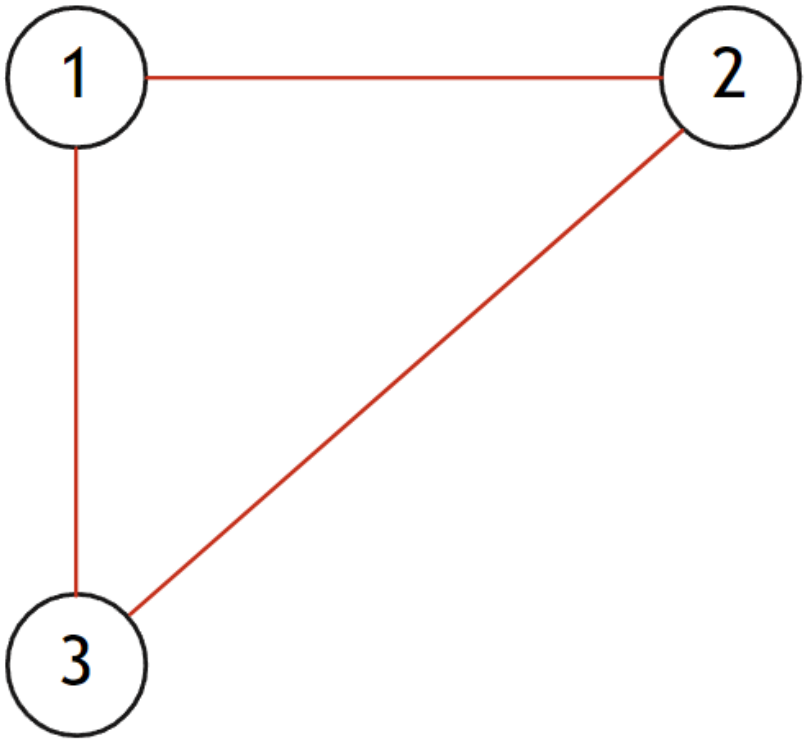


Figure 4.1: Triangle graph

Table 4.1: "Relative time required to calculate the features

Feature	**Time [%]**
degree	1.0
degree centrality	2.0
betweenness centrality	46.7
disjunction value	25.6
random features	2.8
average neighbor degree	6.1
closeness centrality	15.5

which for larger matrices can take a lot of time.

Figure 4.3 shows tests conducted on the test dataset with various used features. As expected, the best-performing model is the one that contains all features. When looking at the model without betweenness centrality, a feature that is the most complex in terms of time, it can be noted that the model still performs well. The same applies to a model trained on data without the disjunction

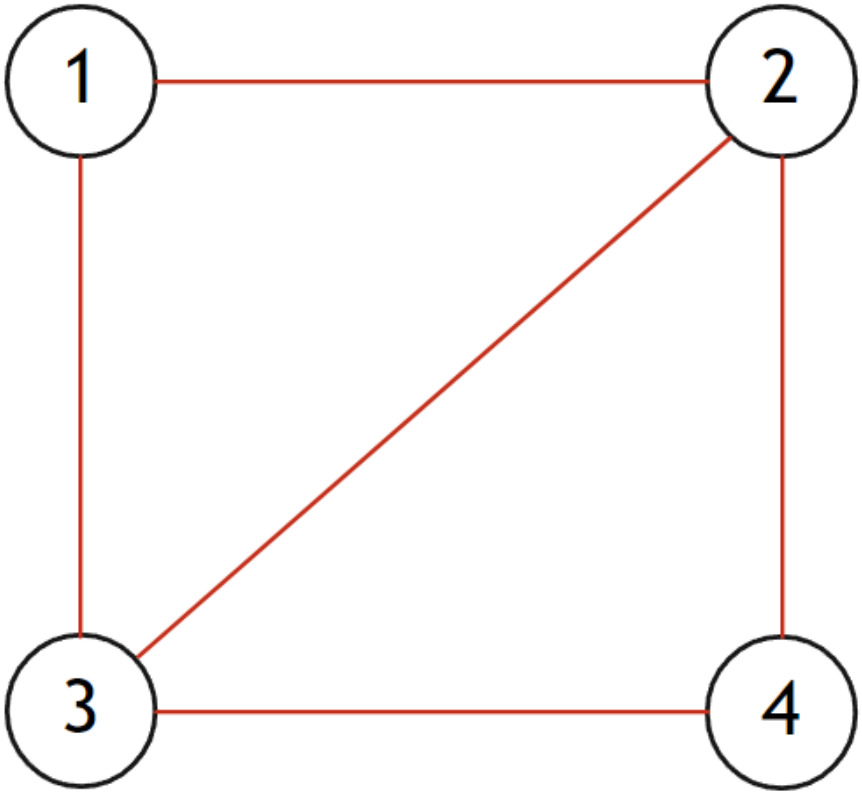


Figure 4.2: Symmetrical graph

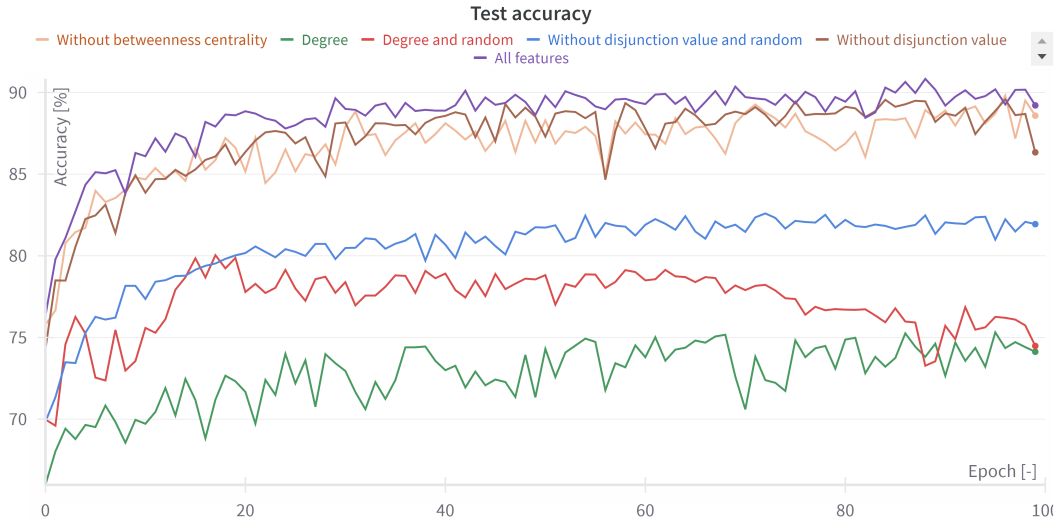
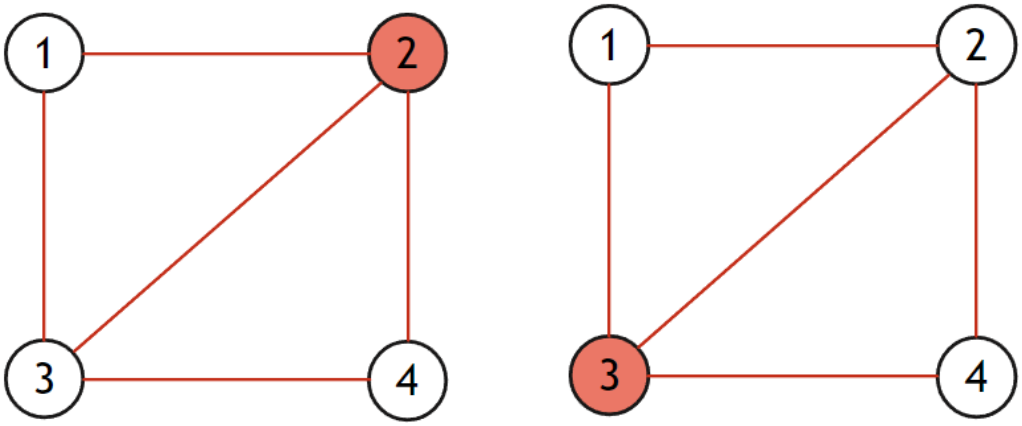


Figure 4.3: Test accuracy with different features

value feature. Comparing this model to the one without disjunctive value and random, it's evident that the random feature provides significant advantages for



(a) One solution **(b)** Other solution

Figure 4.4: MIDS solutions for symmetrical graph

this application. Looking at the simplest models, it can be noted that they perform well even though they are much simpler.

4.4.2. Multiple solutions

As mentioned in Section 4.3, there needs to be an alternative approach for graphs with multiple solutions to the MIDS problem. In this subsection two equally good methods will be presented.

When training a neural network in a supervised manner it is essential to have correct labels which represent true classifications of a certain subject. In this case, we need to distinguish which nodes are a part of MIDS and which are not. Nodes that are part of MIDS are labeled as 1 and those that are not are labeled as 0. If looking at the graph in Figure 4.2, as mentioned there are two possible solutions. One solution is node 2 and the other solution is node 3. These solutions are labeled for use in GNN in the following way:

Figure 4.4a: [0, 1, 0, 0]

Figure 4.4b: [0, 0, 1, 0]

Each field in an array represents one node and the value represents whether it is a part of the MIDS or not.

One method is to code all the solutions into one array with a size equal to the number of nodes. The coding method could vary but the easiest is binary coding. All solutions are arranged in a matrix where rows represent each node in a graph and each column is a solution. The coded solution is then the decimal value of

Table 4.2: Binary coding

Node	Solution 1	Solution 2	Coded solutions
1	0	0	0
2	1	0	2
3	0	1	1
4	0	0	0

such arranged all possible MIDS solutions. Solutions for the graph in Figure 4.2 are listed in Table 4.2 alongside the corresponding coded decimal value. This way, the dataset can be loaded in memory as a batch which makes it more memory efficient. While this approach is usually superior to not using batches, it does not work in this case because of multiple possible solutions to the MIDS problem. Even if the labels are loaded as a batch, during training they would need to be decoded back to binary values. This way the training would be as slow as without using batches.

The other method which is used in this work is to concatenate all the solutions in a row vector. This way it is possible to access all the possible MIDS solutions for a certain graph. If looking at the graph in Figure 4.2, the labels for it would in this case be: $[0,1,0,0,0,0,1,0]$. Knowing the number of nodes in a graph it is easy to split this vector and access each solution individually while training a model. Same as for the previous method, this does not work with batches.

Batch in neural network terms is a hyperparameter that defines the number of examples used in each training iteration (Developers, 2024). In graph problems, each batch includes multiple graphs and their corresponding labels which are interpreted as one bigger graph. Figure 4.5 shows what a batch containing two graphs would look like. A_1 and A_2 are adjacency matrices of two graphs while X_1

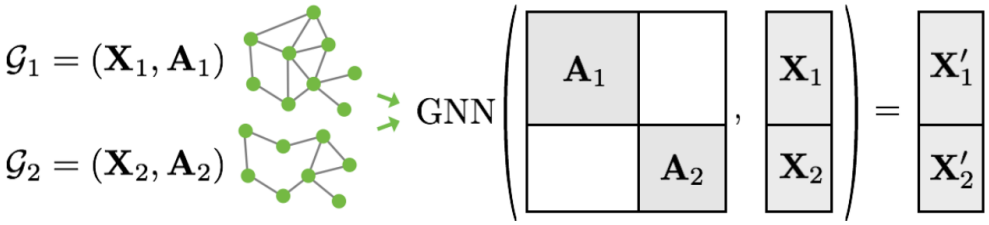


Figure 4.5: Batch of graphs - taken from
https://pytorch-geometric.readthedocs.io/en/latest/get_started/colabs.html

and X_2 are their corresponding features. X'_1 and X'_2 are outputs of a graph neural

network that contain predictions for each node. The reason why this is a faster way of training a GNN model is, as mentioned in Section 4.1, that GPUs are great at computing linear algebra of huge matrices. Because of that it is much more efficient to load one big matrix into the GPU instead of loading each graph as a separate matrix one at a time.

4.5. GNN structure

One of the most important parts of training a neural network model is to choose the adequate architecture. This is not a trivial choice as there are a lot of things to consider. Firstly, the layer types have to be selected. As mentioned in Section 3.2, there are a few popular GNN layers that are most used today. The structure in this work was inspired by Fey and Lenssen (2019) and their example of GNN usage on the PPI dataset[2]. PPI (Protein-Protein Interactions) is a dataset of different protein molecules which are represented as graphs. Considering that the dataset used in this work also consists of multiple graphs, choosing a similar type of structure was implied. The structure utilizes GAT architecture which is somewhat more complex than the others mentioned in Section 3.2. In addition to GAT architecture, each layer contains a simple linear layer that is combined with GAT.

Another parameter crucial for better performance of a neural network model is the number of layers. While in general, each layer of a neural network can contain different architecture, in this work it was constrained that every layer was the same, for simplicity of testing possible architectures. Initially, the number of layers for this task was set to 3 as was in the PPI example, however, while testing having an additional layer endorsed better results. This can be seen in Figure 4.6 where performance for models with different numbers of layers was tested. Additionally, Figure 4.7 shows the time taken throughout training for each of the models. While having 2 layers is the fastest, its performance is substantially worse than others. When looking at all plots on Figure 4.6, the architecture with 4 layers stands out as better than all others. While it is comparable with architecture with 3 layers it still performs better. The only limitation is that it is slower to train so the decision is to be made whether time or accuracy is more important. As seen in Figure 4.7, the time for training an epoch grows linearly with the addition of

[2] $https://github.com/pyg-team/pytorch_geometric/blob/master/examples/ppi.py$

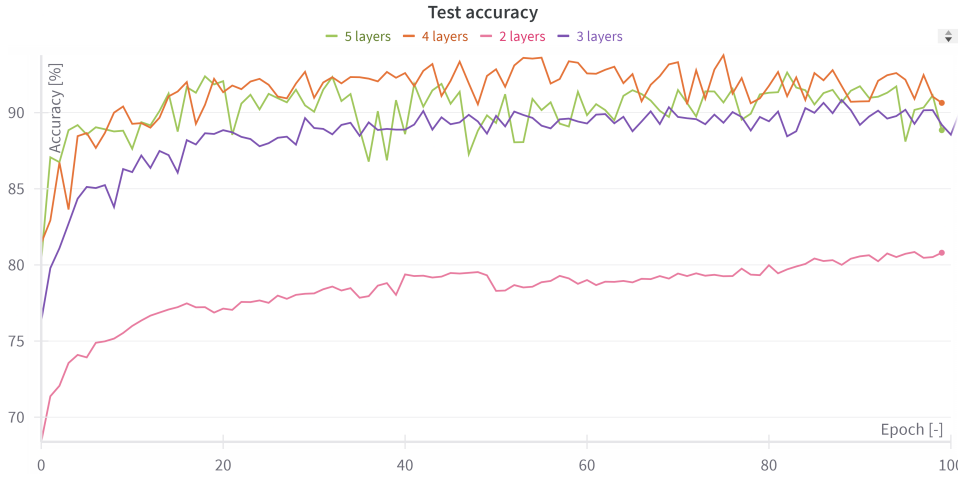


Figure 4.6: Test accuracy for different numbers of GNN layers

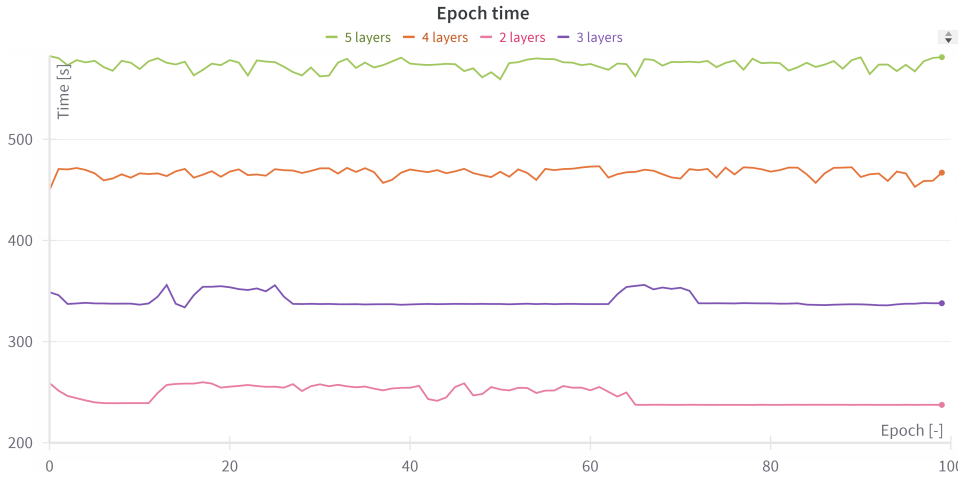


Figure 4.7: Epoch time for different numbers of GNN layers

layers.

Other important parameters to consider are the learning rate and the number of hidden channels. Learning rate, as the name suggests, is a parameter that defines how fast the model trains. In machine learning terms it is a parameter responsible for the rate at which weights of a model change during gradient descent. A bigger learning rate implies more rapid and oscillatory learning while having a smaller learning rate leads to slower but smoother learning. Usually, the learning rate has to be small enough to approach a minimum but large enough so it can escape local minimums and perform faster. The number of channels in each hidden layer defines the dimension of an embedding vector. Having more neurons in theory leads to better performance of a model but that doesn't always have to

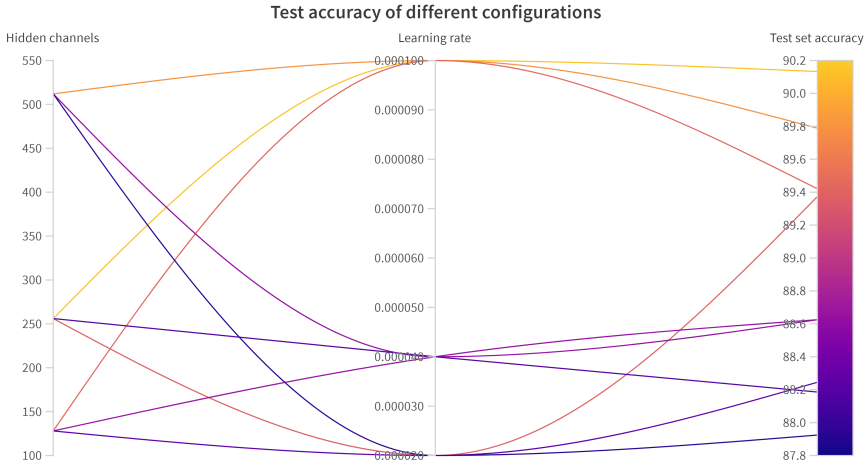


Figure 4.8: Test accuracy with regards to learning rate and number of hidden channels

be true. That is because as model complexity increases it has more weights to train. This can complicate model training and increase the time needed to reach a desired result. Additionally, increasing the number of hidden channels increases training time for each epoch while the learning rate doesn't have much impact on training time. Figure 4.8 shows conducted tests with varying values for learning rate and number of hidden channels. While all models performed in a similar range (88% to 90%), one stood out as the best of all tested configurations. The only model that gave results above 90% on the test set is the one with 256 hidden channels and a learning rate of 0.0001. In Figure 4.8 that model is shown as a yellow line.

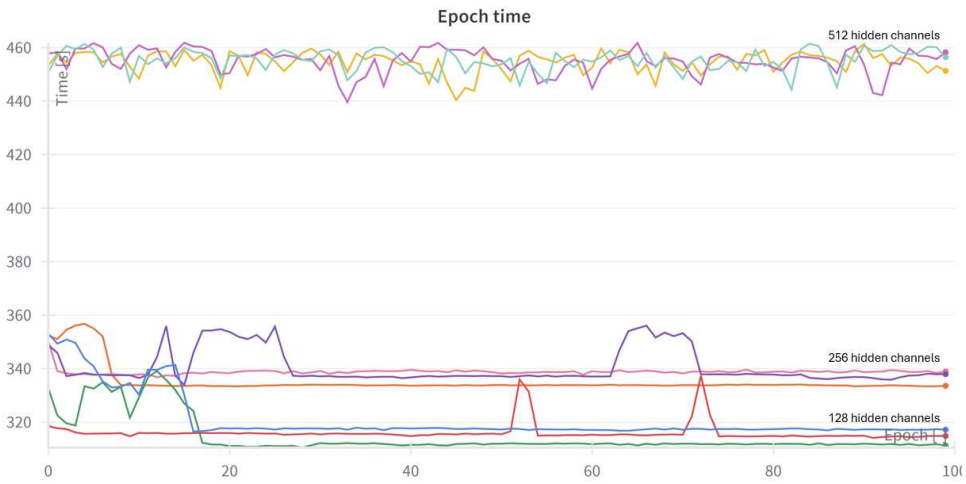


Figure 4.9: Epoch time with regards to learning rate and number of hidden channels

Figure 4.9 shows the time taken for one epoch for each of the models shown in Figure 4.8. As previously said, a higher number of hidden channels increases epoch time while the learning rate doesn't have much impact. In the figure, there are 3 separate groups of models. In each of 3 groups learning rate values are 0.00002, 0.00004, and 0.0001. The group of models that have 128 hidden channels is the one around 320 seconds, models containing 256 hidden channels have an epoch time of around 340 seconds and the others (which are around 450 seconds) are the ones with 512 hidden channels. When looking at the plots there is a clear separation between models containing 512 hidden channels and others. From this experiment, it can be concluded that having 512 hidden channels is too time-consuming.

4.6. Training and testing

4.6.1. Training

Training is the core of every successful neural network model. Training a model is a term that is used for describing a process of adjusting weights based on the current model output and current weights of a system. As mentioned in Section 4.4, it is important to have a large number of examples in a training dataset. Those examples are crucial as by comparing model output and each example, the measure is formed which tells a model how wrong its prediction is. This is done in every iteration and that measure is called training loss. Having a training loss is just a part of the training process. For adjusting the weights of a model an optimizer is needed. Optimizer is a specialized function for calculating the gradient descent.

In this work, optimizer Adam was used which is one of the most popular optimizers nowadays. This optimizer is an extension of a stochastic gradient descent which was introduced by Kingma and Ba (2017). As for the loss function, none of the existing ones could be implemented for this problem. As mentioned in Section 4.4.2, this dataset contains labels configured in a peculiar way. For that reason, it was necessary to formulate a custom loss function that can handle data formulated in such a way. The loss function is based on the function BCEWithLogitsLoss. This function is a combination of BCELoss and a sigmoid function. BCELoss (Binary Cross Entropy Loss) is a cross-entropy loss function that is specialized for working with binary classification (Ansel et al., 2024). To implement and reconfigure this function it is needed to override its $forward()$

function. The function is shown on Listing 4.1. The function is constructed in a way to iterate over all possible correct MIDS solutions and preserve the loss which is the smallest. In this way, it is not limited to one solution and for one training iteration, it considers just the one that is the closest to any MIDS solution. In the function, variable *input* represents a tensor that contains a prediction and *target* is an array that contains all solutions in one row array. Because of the chosen way in which the dataset labels are formulated, this array of solutions needs to be split into multiple arrays that have the same size as the number of nodes in the graph. After this is done it is possible to perform the algorithm which returns the minimum loss for a given graph.

Listing 4.1: Custom Loss Function in PyTorch

```
class CustomLossFunction(torch.nn.BCEWithLogitsLoss):
    """Custom loss function based on BCEWithLogitsLoss"""
    def forward(self, input: Tensor, target: Tensor) -> Tensor:
        tens = torch.split(target, input.size(dim=0))[0]
        loss = F.binary_cross_entropy_with_logits(input, tens, self.weight,
            pos_weight=self.pos_weight, reduction=self.reduction)

        for tens in torch.split(target, input.size(dim=0)):
            new = F.binary_cross_entropy_with_logits(input, tens, self.weight,
                pos_weight=self.pos_weight, reduction=self.reduction)
            if new.item() < loss.item():
                loss = new
        return loss
```

4.6.2. Testing

To know if a model is trained correctly it needs to be tested. Testing can be performed in different ways and is used to evaluate the performance of a model. The dataset is usually divided into one for training and the other for testing. A training dataset is used for training a model while a test dataset is composed of data that the model does not use for training. Having a test dataset is essential for determining whether the model performs well on just training data or all data as wanted. In other words, does the model have the ability to generalize on previously unseen data? The most common testing measure is loss, calculated during training for each example. This value is the average loss across all examples in a dataset. This value is expected to approach zero at infinity for the training dataset, but for the testing dataset, it should have a minimum value, which is the optimal solution. Figure (4.10) shows one such example when training a GNN model. The testing loss value (blue) is higher than the training one (red), as

expected. Another and more interesting testing measure is the testing accuracy.

Figure 4.10: Loss function plot

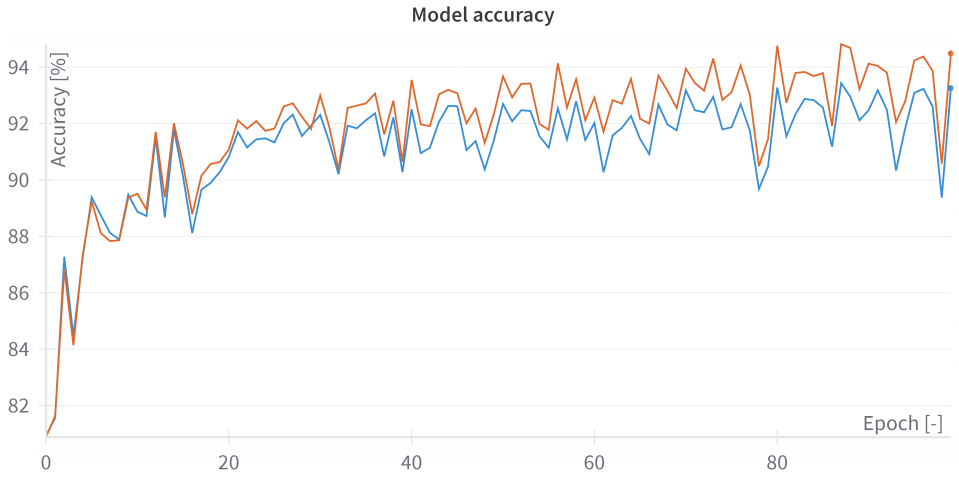


Figure 4.11: Accuracy plot

That is a value that shows how correct the model is. To explain it differently, it tells how accurately the model makes the correct prediction. This is usually a value that is given as a percentage across a dataset. While the training accuracy is also important, the testing accuracy is the one that is used for the correctness of a model. Figure 4.11 shows a plot of accuracy throughout the training of one of the models. The training accuracy (red) reaches higher accuracy levels than the testing one (blue). This is expected because the model hasn't seen examples from the testing dataset during the model training. To test the accuracy of each prediction it is necessary to check whether it is MIDS or not. This can be done in

various ways, but certain criteria need to be met for a set of nodes to be MIDS. The method implemented in this work is shown in Listing 4.2. Firstly, the set of predicted nodes has to be the same size as the MIDS size. Secondly, the set must be independent and dominating. If those criteria are met then the function returns ***True***, if any condition is not met it returns ***False***.

Listing 4.2: Function for checking if set of nodes is MIDS

```
def check_MIDS(A, candidate, target_value):
    n = len(candidate)

    # Candidate set is larger than MIDS size
    if sum(candidate) > target_value:
        return False

    # Candidate set is not dominating
    if not all((A + np.eye(n)) @ candidate >= 1):
        return False

    # Candidate set is not independent
    for i in range(n):
        for j in range(i+1, n):
            if candidate[i] and candidate[j] and A[i,j]:
                return False

    #This case should never happen
    if sum(candidate) < target_value:
        print(f"Somehow we found an even smaller MIDS: {sum(candidate)}, {
            target_value}")

    return True
```

5. Results

To determine if a model is successful it has to be tested on data it hasn't seen before. In training, the dataset is split into train and test datasets. In those datasets, the examples are similar. In this case, they contain graphs of the same sizes. The accuracy of a testing set is a great measure of the model's performance.

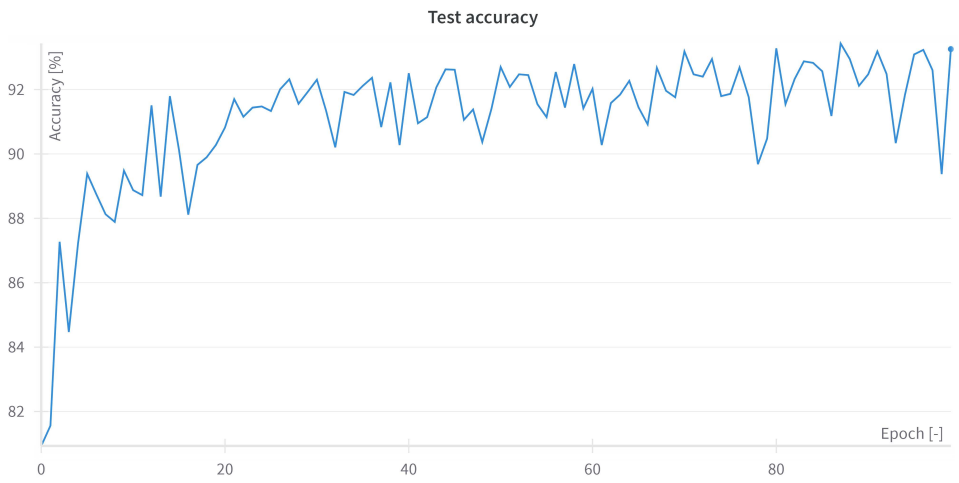


Figure 5.1: Test accuracy plot

Figure 5.1 shows the best-performing model's accuracy. This model has the following structure:

Architecture - GAT + Linear

Number of layers - 4

Features - all

Number of hidden channels - 256

Learning rate - 0.0001

Number of graphs - 42,111

The peak accuracy of this model on the testing dataset is 93.28%. That means the model correctly determines the MIDS solution on 93.28% of selected graphs. This accuracy was determined only by knowing the model output. If for a certain node, the value of an output is positive that node is classified as a part of the MIDS. However, there is an improvement to this measure. The output of the model gives the certainty that a node is a part of the MIDS or it isn't. This can help other methods find the correct solution faster. Knowing the MIDS size of a selected graph, the model's output can be interpreted differently. By combining those values, the prediction can be made by selecting the largest values from the output. Let's look at the example where the model output is [-0.9, -0.1, -0.4, 0.5, -0.2]. The first measure which looks at just the model output would conclude that the MIDS is [0, 0, 0, 1, 0]. However, knowing that the MIDS size must be 2, the second measure would determine the MIDS is [0, 1, 0, 1, 0]. Having that knowledge the predictions become even more accurate.

5.1. Generalization

To test if a model can generalize even more, it is necessary to perform tests on larger graphs. Generalization is a property of a model to make correct predictions on data it has never seen before. As mentioned before, the model can generalize on data from the testing dataset, but it is also important to test its performance on graphs with more nodes. The model was trained on data up to 9 nodes and

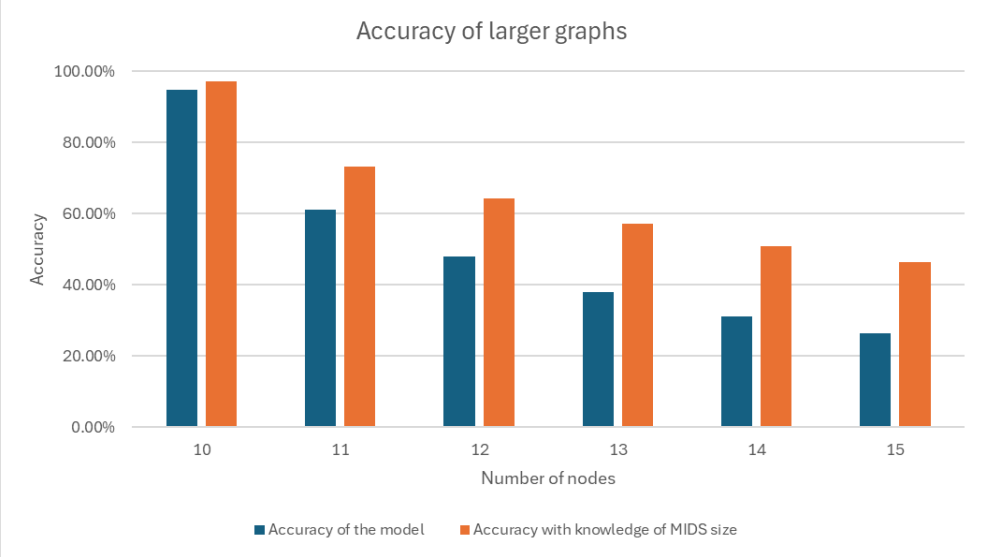


Figure 5.2: Test accuracy plot

has never seen larger graphs. Figure 5.2 shows the accuracy of the model on graphs from 10 to 15 nodes. For each size, 10000 random graphs were used and both interpretations of the model output are shown. As expected, the larger the graphs get the less accurate the prediction is. However, when looking at output interpretations, it is clear that the second interpretation works better. On larger graphs, the difference is even clearer. When looking at the accuracy of graphs with 15 nodes, the model output prediction accuracy is 26.27% while the accuracy with knowledge of the MIDS size is 46.34%. That is almost twice as good as just the model output.

6. Conclusion

To conclude, in this thesis, the task was to train a GNN model to be able to determine the solution to the MIDS problem. MIDS is a complex combinatorial problem that has deterministic solutions in exponential time. Neural networks, and specifically graph neural networks can help solve combinatorial problems in graphs. The GNN model was trained on graphs up to 9 nodes. We showed that MIDS can be found with great accuracy on graphs of up to 10 nodes. Increasing the number of nodes in testing graphs results in less accurate results. The complexity of the model can be decreased by excluding the most complex features. This comes at a cost of accuracy but the dataset generation and model training would be faster. The results shown in this thesis are promising and could possibly be improved by implementing different GNN architectures or utilizing different features.

Bibliography

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, April 2024. doi: 10.1145/3620665.3640366. URL `https://pytorch.org/assets/pytorch2-2.pdf`.

Lukas Biewald. Experiment tracking with weights and biases, 2020. URL `https://www.wandb.com/`. Software available from wandb.com.

Christian Blum, Borja Calvo, and Maria J. Blesa. FrogCOL and FrogMIS: New decentralized algorithms for finding large independent sets in graphs. *Swarm Intelligence 2015*, 9(2):205–227, July 2015. ISSN 1935-3820. doi: 10.1007/S11721-015-0110-1.

Ameya Daigavane, Balaraman Ravindran, and Gaurav Aggarwal. Understanding convolutions on graphs. *Distill*, 2021. doi: 10.23915/distill.00032. https://distill.pub/2021/understanding-gnns.

Google Developers. Machine learning glossary, 2024. URL `https://developers.google.com/machine-learning/glossary`.

J. Domislović. Određivanje minimalnog nezavisnog dominantnog skupa u općenitim grafovima. Master's thesis, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, Zagreb, 2022. URL https://urn.nsk.hr/urn:nbn:hr:168:624118.

Chandra Eksombatchai, Pranav Jindal, Jure Z Liu, Ying Liu, Rahul Sharma, Charles Sugnet, Michael Ulrich, and Jure Leskovec. Pixie: A system for recommending 3+ billion items to 200+ million users in real-time. In *Proceedings of the 2017 World Wide Web Conference (WWW)*, pages 1775–1784, 2017. doi: 10.1145/3038912.3052715.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Serge Gaspers and Mathieu Liedloff. A branch-and-reduce algorithm for finding a minimum independent dominating set. *Discrete Mathematics & Theoretical Computer Science*, 14(Graph and Algorithms), 2012.

Fabrizio Grandoni. A note on the complexity of minimum dominating set. *Journal of Discrete Algorithms*, 4(2):209–214, 2006. ISSN 1570-8667. doi: https://doi.org/10.1016/j.jda.2005.03.002. URL https://www.sciencedirect.com/science/article/pii/S1570866705000225.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf.

Kazuya Haraguchi. An Efficient Local Search for the Minimum Independent Dominating Set Problem. *Leibniz International Proceedings in Informatics, LIPIcs*, 103, February 2018.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Federico Monti, Fabrizio Frasca, Davide Eynard, Damon Mannion, and Michael M. Bronstein. Fake news detection on social media using geometric deep learning, 2019. URL https://arxiv.org/abs/1902.06673.

Kyoung-Su Oh and Keechul Jung. Gpu implementation of neural networks. *Pattern Recognition*, 37(6):1311–1314, 2004. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2004.01.013. URL https://www.sciencedirect.com/science/article/pii/S0031320304000524.

Thomas Pontoizeau, Florian Sikora, Florian Yger, and Tristan Cazenave. Neural Maximum Independent Set. *Communications in Computer and Information Science*, 1524 CCIS:223–237, 2021. ISSN 18650937. doi: 10.1007/978-3-030-93736-2_18/FIGURES/7.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W Battaglia. Learning to simulate complex physics with graph networks. *arXiv preprint arXiv:2002.09405*, 2020.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 333–341, 2021. doi: 10.1137/1.9781611976700.38. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611976700.38.

Chao Shen and Tao Li. Multi-Document Summarization via the Minimum Dominating Set. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 984–992, USA, 2010. Association for Computational Linguistics.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

Jie Wang, Lei Zhang, Wenda Zhu, Yuhang Jiang, Wenmin Wu, Xipeng Xu, and Dawei Zhao. The Suppression of Epidemic Spreading Through Minimum Dominating Set. *Frontiers in Physics*, 0:525, January 2021. ISSN 2296-424X. doi: 10.3389/FPHY.2020.588513.

Yiyuan Wang, Ruizhi Li, Yupeng Zhou, and Minghao Yin. A path cost-based grasp for minimum independent dominating set problem. *Neural Computing and Applications*, 28, 12 2017. doi: 10.1007/s00521-016-2324-6.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=ryGs6iA5Km`.

Određivanje minimalnog nezavisnog dominantnog skupa primjenom grafovskih neuronskih mreža

Sažetak

U ovome diplomskom radu razvijena je metoda za određivanje minimalnog nezavisnog dominantnog skupa (MIDS) u općenitim grafovima zasnovana na grafovskim neuronskim mrežama (GNN). Generiran je skup grafova s pripadajućim MIDS rješenjima koji je služio za treniranje modela GNN-a. Skupu grafova pridružene su i odabrane pripadajuće značajke za svaki čvor koje mogu na jedinstven način opisati graf. GNN model treniran je na s različitim konfiguracijama kako bi se pronašla ona s najboljim rezultatima. Na najboljem modelu zapaženi su izvrsni rezultati na testnim podatcima s točnošću od preko 93%. Iako je model treniran na grafovima s do 9 čvorova, može i generalizirati na većim grafovima. Povećanjem broja čvorova na grafu točnost pronalaženja rješenja MIDS problema pada. Ovim radom pokazana je mogućnost rješavanja MIDS problema primjenom GNN modela.

Ključne riječi: MIDS, GNN, grafovi, neuronske mreže

Minimum independent dominating set calculation based on graph neural networks

Abstract

In this thesis, a method for determining the minimum independent dominant set (MIDS) in general graphs based on graph neural networks (GNN) was developed. A dataset of graphs with associated MIDS solutions was generated and used to train the GNN model. Each node in generated graphs was associated with a set of features that together uniquely described each graph. The GNN model was trained on different configurations to find the one with the best results. The best model demonstrated excellent results on the test dataset with an accuracy of over 93%. Although the model is trained on graphs with up to 9 nodes, it can also generalize on larger graphs. By increasing the number of nodes on the graph, the accuracy of finding a solution to the MIDS problem decreases. This work demonstrated the possibility of solving the MIDS problem using the GNN model.

Keywords: MIDS, GNN, graphs, neural networks