University of Bergen
Department of Informatics

Solving Maximum Weighted Matching problem using Graph Neural Networks

Author: Nikita Zaicev
Supervisor: Fredrik Manne

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

March, 2024

Abstract

In this work we tried to train a Graph Neural Network (GNN) to solve the Maximum Weighted Matching problem on graphs.

Acknowledgements

I want to thank Fredrikk Manne, Kenneth Langedal and Johannes Langguth for helping me with this work.

Nikita Zaicev
Sunday 10th March, 2024

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

This chapter is an introduction that explains the goals and reasons for this work.

1.0.1 Goal

Machine Learning and Neural Networks have shown to be extremely potent and versatile in solving vast variety of problems across different fields. One research field that has been popular and challenging in the last few years is Combinatorial Optimization (CO). CO includes problems such as Maximal Indepndent Set (MIS) and Maximum Weighted Matching (MWM). Such problems can be viewed in the context of graphs. GNN is a subclass of Neural Networks designed specifically for solving problems related to graphs, but there are some problems that have not yet been solved efficiently with GNNs.

The main goal for this project is to: "Find out whether a GNN can outperform approximations algorithms such as greedy in solving MWM problem.

1.0.2 Reason

Previous section bescribes "what" is this work about. Now lets breifly discuss "why" do this in the first place. The idea is rather simple. There are 2 types of algorithms in general: exact and approximate. The exact algorithm for MWM already exists, but the down side of such algorithms is their speed. The slower nature of exact algorithms is one of the reasons why approximation algorithms are used in some cases. The hope is

that GNN can potantialy squeeze between the approximation and optimal algorithms in terms of both time and accuracy.

In short the reason is: "There can be a machine learning based algorithm that gives better results than existing approximation algorithms"

1.1 Project structure

The rest of this document has the following structure:

1. Background:

   This Chapter will briefly touch the history, applications and impact of machine learning and go through core concepts and ideas behind machine learning and neural networks. Relevant information about CO and relevant algorithms will also be included here.

2. Methodology and Data:

   This Chapter will focus on how the research was done and also explain core concepts of Neural Networks and the general procedure of training a Neural Network. Then the specifics of this case will be discussed together with the main challenges. The progress made step by step and the reasoning behind changes and choices made along the way will be shown. The chapter will also analyze the data used for training the model and evaluating results along with justification for the chosen data.

3. Results:

   This Chapter will focus on temporary results produced during the reseach and explain how theese results affected the further decisions. Finally the chapter will be concluded by analyzing the final results and comparing them with the expactations.

4. Conclusion:

   Here the conclusion for this project will be drawn regarding whether the results gave any meaningfull insight and what future work can be done for improvements.

1.1.1 Listings

You can do listings, like in Listing 1.1

Listing 1.1: Look at this cool listing. Find the rest in Appendix A.1

```
$ java -jar myAwesomeCode.jar
```

You can also do language highlighting for instance with Golang: And in line 6 of Listing 1.2 you can see that we can ref to lines in listings.

Listing 1.2: Hello world in Golang

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("hello world")
7 }
```

1.1.2 Git

The whole project can be seen here: `https://github.com/nikitazaicev/Master`

Chapter 2

Background

This chapter explain all the relevant concepts, technologies and problems relevant to this project.

2.1 Neural Networks and Machine Learning

Machine learning in computer science is a broad term for all the algorithms that using statistical methods try to learn patterns and make predictions. Some of the earliest algorithms were invented already in the 1940s [5], but it is notably in recent times, with increasing computational power of the computers and GPU's, that machine learning and especially neural networks have shown impressive results in a variety of fields such as recognition and generation of images, videos, sound and text (references?).

Machine learning can be further split into supervised and unsupervised learning. Supervised learning means the model is given correct answers it can learn from by trial and error and then apply gained knowledge on the useen data. The other option is unsupervised learning where the model usually has some formula it can use to calculate how far it is from the set goal. In this case we will be trying supervised learning where we solve MWM problems and use them to traing the model.

Neural networks are a subclass of machine learning algorithms that are inspired by the human brain. They are for the most part made of different layers that consist of "neurons". Layers can be grouped in 3 types: an input layer that reads the given data, one or more middle layers that proccess the data and output layer that give the final

answer. Each neuron in a layer is usually connected to all the neurons of the next layer, although it may depend on the architecture. Between connected neurons there are weights. When the neural network model recieves input in is passed through the layers and gets recalculated using the weights. The result at the output layer is some altered numerical representation of the input that can, depending on the goal, be interpreted as some prediction, for example a probability. The weights of neural network can then be adjusted depending on the accuracy of the prediction. After the adjustment is done model can try to make a prediction again. This process can be repeated multiple times, until it the model can make predictions well enough. This process is called training. An important thing to keep in mind is that a trained model needs to be tested on unseen data since it could have just memorized all the data it was trained on to make predictions. In machine learning one is interested in generalization, the model needs to find some patterns that are common for all the relevant data it can encounter.

In this case the model will be trained using

2.2 Graph Neural Netoworks

GNN is a relatively young subclass of neural networks designed specifically for solving graph related problems and it has been a growing topic of research in the last couple of years.

Lorenzo Brusca and Lars C. P. M. Quaedvlieg et al. [2] showed a self-training GNN for MIS.

Schuetz et. al. made an unsupervised GNN [7] for solving MIS and Angelini and Ricci-Tersenghi [1] compared its performance to greedy algorithms and reported some problems with GNNs performance.

The reason MIS solving is mentioned so much here is because the problems are to some degree related, as algorithmic problems often are. There will be concrete examples in the Methodology and Data chapter. MIS is also more often a subject to research and researches on MWM are not that common. Bohao Wu and Lingli Li tried to solve MWM using deep reinforcement learning [9] and reporting that "Experimental results show that L2M outperforms state-of-the-art algorithms."

GNNs are based on the same concepts as the neural networks, but with some modifications to better suit graph related problems. Opposite to the more standart neural

networks, GNN makes a representation of each node based on its neighbors. Adding additional layers increases the depth of neighborhood used. Meaning if a model has 2 layers the representation of one node would be calculated from nodes neighbors and the neighbors of the neighbors. This allows the GNN to utilize graph structure unlike a standart neural network.

2.3 Maximum Weighted Matching and Combinatorial Optimization

Combinatorial optimization is a field of study that covers problems that require finding a subset or a combination from some set of elements, that fulfills certain requirements. An example of such problem would be Maximum Weighted Matching and Maximum Independent Set.

MWM problem asks to find a subset of node pairs in a weighted graph, such that sum of weights is maximized. The problem is relatively simple compared to many others is the field of CO and has optimal algorithms that run in polynomial time such as blossom algorithm [4]. Optimal algorithms however can still be rather slow when working with large datasets, and in some cases approximation algorithm may be more of a suitable solution if the exact answer is not critically important. The most simple example is a greedy algorithm that sorts all the edges by their weight and picks them in descending order.

The MIS asks to find a subset of non-adjecent nodes such that the total amount of nodes is as large as possible. There is a similar problem Maximum Weighted Indepndent Set (MWIS) which asks to find an independent set that gives the largest total weight sum. MWIS is partially relevant as we will later try a graph transformation that essentialy turn the original MWM problem into MWIS.

2.4 Solving Combinatorial Optimization Problems with Graph neural Networks

Chapter 3

Research Methodology and Data

This chapter focuses on describing which approaches were chosen for the task at hand and why, as well as shows how the work progressed.

To reiterate the current problem is to find out if GNN can compete at solving the MWM problem compared to other approximation methods such as greedy algorithm.

3.1 Challenges

It is common to think of a neural network as of a black box where you give some data to this box and it tries to predict the correct answer. In this case the data is a graph consisting of vertices connected by edges that have weights and the expected answer should be pairs of vertices that were matched together.

As mentioned the idea behind supervised learning is to give a model the correct answers so it can by trial and error learn from it. Theese answers are not included in the initial datasets so the optimal solutions needs to be calculated. To find the optimal solution Blossom algorithm implemented by Joris van Rantwijk in Python was used [8]. This does however point out an important weakness of the supervised approach. Obviously a model needs to be able to handle graphs of different sizes and it is the large ones that are most interesting. Finding an optimal for algorithmic problems for the large graphs can be time consuming, at the same time a model need as much data as possible to learn leading to multiple large graphs consuming to much time. This poses a question

whether it is possible for the model to learn on small and medium sized graphs that are not as time consuming and transfer learned patterns to solve larger graphs.

Another important aspect that must be taken in to the consideration is that it is unlikely for the model to fully follow the restictions of the problem. Model might decide to match same node to 2 neighbors at the same time, which should not be allowed.

3.2 Main Idea

To summarize, given a weighted udirected graph, the Neural Network must predict a valid subset of edges or in other words pairs of vertices that maximizes the total weight. The Neural Network will have a graph as an input and the otput will be the probabilities (0.00 - 1.00) of how likely an edge will be in the matching. To control that a solution is valid, instead of picking all the edges with the probability higher than 50%, the edges can be sorted by their probabilities and greedily picked in that order if they don't break the validity of the solution. This might sound illogical to use greedy algorithm on the models output to beat the normal greedy algorithm with weights, but the fact that edges are now sorted not by their weight, but a score decided by Neural Network does make a difference, since the GNN can take multiple features in consideration when assigning probabilities. Some of theese features come naturaly such as structure of the graph, and some we can manualy add during preprocessing.

Neural Networks have a lot of hyperparameters and methods that can be tuned to make a model better suited for the task. The ideal way of finding the best combination of the hyperparameters is to try as many combinations as possible, then train a model for each combination and choose the one with best performance. This is a time consuming procedure however and therefore in this task the exploration of hyperparameters was narrowed down to the ones that seemed most important and indicated positive impact during early experiments.

During the experiments following hyperparameters were tested:

1. Learning rate 2. Class weights 3. Weight decay 4. Network depth and width (layers and neurons)

5. Additionally other methods were tried such as augmenting data (adding extra features to the nodes during preproccessing): 5.1. degree 5.2. weights relative to the neighbors 5.3. difference between weights of the neighbors 5.4. sums of the weights. Also for the edge classification 1st largest weight, 2nd largest weight were added to match the reduction rule that was tested later.

3.3 Data

The model should be capable of solving any graph relevant to MWM problem. A relevant graph can be difined by following characterisitcs Ideally the model should be able to handle any kind of an udirected graph with weighted edges.

For training and experimenting with the GNN, a MNIST dataset was used [3]. The dataset consists of 70000 relatively small graphs with 70 nodes and 564 edges on average. Graphs also include features for the edges representing distances between node. Theese features is what is used as the weights for the problem. Although the graphs in this dataset are realatively small they have some fitting qualities such and nodes having many neighbors creating more possible ways to match the nodes. Small size also makes it less time consuming to train the models and try different approaches as well as shows whether a GNN trained on smaller graphs can transfer its knowledge to larger graphs.

A couple large graphs from SuiteSparse were chosen for testing performance on large graphs.

Small handcrafted graphs were used to test if GNN can at least beat the cases specificaly made to abuse weaknesses of a greedy approach.

3.4 Architecture

3.4.1 Data preproccessing

About transforming to linegraph, additional features, trying reduction rules and normalizing weights...

3.4.2 Model pipeline

About multiple iterations of passing graph through the model in case not the whole graph is matched and cleaning the remainder using normal greedy.

3.4.3 Line Graph Approach

Line graph approach was the first attempt at using a simple GNN, which later turned out to be too time consuming for larger graphs to be worth further expriements. It did however give some usefull insight as well as a showed to be a proof of concept. In the context of graphs line graph is a complement of the original graph that turns each edge to a vertex and connects the vertices if they shared a vertex in the original graph.

Examaple of a graph and its line graph convertion

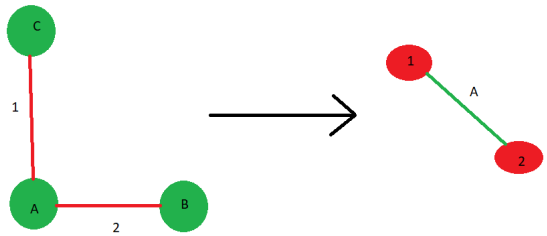

Figure 3.1: Original graph (left) and its line graph (right)

Converting to a line graph makes the architecture of the GNN model simpler. Instead of needing to ask model for each pair of nodes is they should be matched together, the line graph model now can output the probability of a node being in the matching directly, since the node now represents two connected nodes from original graph. This also however transforms the problem itself. From the perspective og the model it is now trying to solve MWIS. GNNs for MIS have been studied before. Nouranizadeh et. al. showed demonstrated a pooling method for solving MWIS [6].

3.4.4 Edge Classification Approach

A more natural way of approaching this problem is for each pair of vertices that are conneced by an edge, ask the model if given pair should be a part of the matching. In other words model is classifying an edge.

3.5 Result Validation

Before looking at the results it is important to decide how to evalueate results properly and what is important for the problem at hand:

1. Time - how long an algorithm took to produce an answer.
2. Correctness - is the answer correct. In case of MWM the total weight aquired would be the measurement of how correct the solution is. It is unlikely that GNN can find an optimal solution for more complex problems so it is reasonable to look at how close GNN comes to the optimal solution.
3. GNN portion - due to the way program is set up, if the model does not match the whole graph, the rest of the graph will need to be finished somehow. This is done by running normal greedy algorithm at the end if anything is left. Because of that there can be cases where the model does nothing and by default achieves 100% result of what normal greedy algorithm would. Therefore the portion of the weights obtained directly by the model needs to be evaluated as well.

3.5.1 Sanity checks

It can often happen that during the project something goes wrong with the model and it can be not obvious. Therefore, one often does what is called sanity checks to see if model's inputs and outputs still make sense. Following was done to ensure the model and data are functioning correctly:

1. Train model on a small dataset a lot and test it on the same data. The model should be able to memorize theese graphs and solve the problem near perfect on them.

3.6 Expected Results

3.6.1 Accuracy and total weight

There are not that many researches specifically for MWM, but problems like MIS that are relatively close to MWM can indicate similar results for this case as well. As discussed

in the Background section, there are researches that show that GNNs are capable of solving CO problems and beating greedy algorithms, while other rather indicate that improvements are still needed for it to be worth using. Therefore it is hard to forecast any results based on previous work. Nothing stands in the way of being optimistic however, additionally to the fact that greedy algorithm is relatively simple and Neural Network should be able the recognise a more complex pattern it can use to achieve better results. It is absolutely not expected for the model to be able to find optimal solution since any Neural Network is a heuristic. The margin by which GNN can surpass the greedy solution is expected to be rather small, since from the data analysis it was abserved that for the majority of graphs greedy algorithm preforms rather well with above 80% of the optimal possible weight.

3.6.2 Time

GNN model is a heuristic solver and gives an approximate answer. Therefore model should be noticeably faster than exact algorithm, otherwise it would not be worth it. The time a model takes to solve one instance of a problem should be closer to that of a greedy algorithm and probably slightly longer due to preproccessing required such as augmenting data with adittional features. Naturaly, time will also depend on the depth and the width of the network.

3.6.3 Remainder or model portion

Ideally the model should be able to handle the full graph on its own, but as long as the model manages to improve final result it would still be a usefull tool. A speculative estimation for the sake of having something for orientation will be 80%. That is the weight obtained from models prediction makes at least 80% of total weight.

3.6.4 Other observations

One improtant observation was made during the analysis of the data that is worth mentioning. The difference between the optimal solution and the greedy is in most cases is rather small. For the MNIST dataset and the majority of the graphs found on Suite Sparse Matrix Collection the greedy algorithm manages to reach 90+% of the optimal weight. Assigning random evenly distributed values to the edges weights also seem to give the same results. For the GNN this means that it will be rather difficult to beat the greedy algorithm since it gets very close to the optimal.

Chapter 4

Results

This chapter lists the temporary results from the experiments and thoughts on what it meant for the model, as well as the final results. All the experiments have been done on the same machine with: 11th Gen Intel(R) Core(TM) i7-11700K 3.60GHz 8-core CPU, 16 GB RAM and NVIDIA RTX 3080Ti graphics card.

4.1 Progress

In this section we will go throught the whole proccess step by step, from first iterations of the very simple model to more fine tuned models and other methods that were tested.

4.1.1 Simple line graph model

First step was to try the simplest model with most of the hyperparameters set to default and train it on a smaller scale. First model had 2 layers and 64 neurons each. Initially due to majority of nodes being not included in the matching, the model learned to set all nodes to be dropped and still get a rather high accuracy score. This was resolved by adding class weights as a training parameter. Class weights tell the model how important each class is. By assigning the nodes that go into matching a value of 0.9 and the ones that do not 0.1. The problem seemed to be resolved.

Trained on a 1000 MNIST graphs model resulted on average with 55% of optimal weight possible. This was an expected result considering the limitation, but it at least

showed that model is gaining some knowledge compared to untrained model as well as a solution that randomly picks edges, where both resulted in 50-51% of optimal.

A model that barely beats random solution is not very usefull. Poor performance for now is mainly to the very limited training, but before adding more data and prolonging training, we can experiments with the parameters of the network to see is they make any impact.

4.1.2 Model improvements and data augmentation

It would make sense to start with the depth and with of the network. After trying to add up to 6 layers in general it showed more than 3 layers did not have any significant effect. Note: layers tell how many generations of neighbors is used for representation of a node. Increasing breadth did help, but impact got smaller as the breadth got wider. It would also impact time consumptions so it was decided to have the model with 2 layers and 640 nodes at each layer (except input/output layers). Results were, however still rather low at 58%.

Next step was to adjust optimizer parameters: learning rate and weight decay.

Too high learing rates resulted in too unstable loss function, too low learning rate did not give enough progress and made training proccess too slow. Value of 0.001 worked well as middle ground

Weight decay

5. Trying skip connections, but not noticing any effect.

6. Augmenting node features. Trying each additional feature separately to see it they have any benefits by themselves, then adding them one by one. Finally using all at once since they can mostly be calculated simultaneously and all have at elast some positive effect:

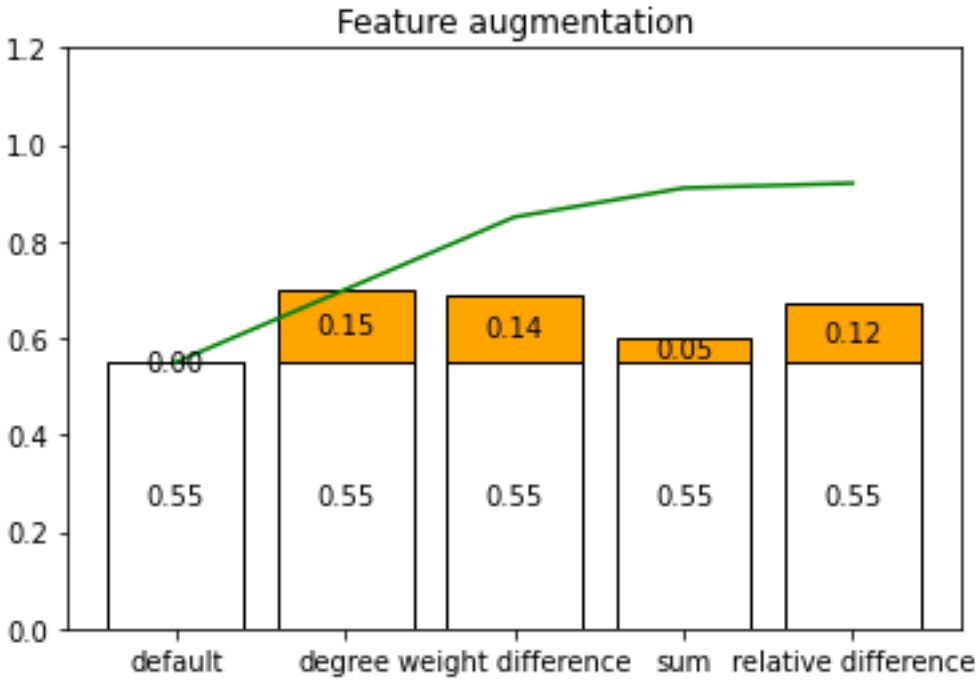

Figure 4.1: MNIST trained model. Average performance on 100 MNIST graphs

With results starting to move closer to reasonable, now larger training set could be used. Unfortunately line graph transformation on a dense graph turned out to expode in size and take to much time to proccess. Therefore another architecture was considered. Trying edge classification. With same hyperparameters and repurposed node feature augmentation.

4.2 Edge prediction model

1. Edge classification is a little bit worse on average. In theory it can be due to line graph containing more structural information in it compared to original. 0.88% of greedy

MNIST trained Model's performance on unseen MNIST graphs

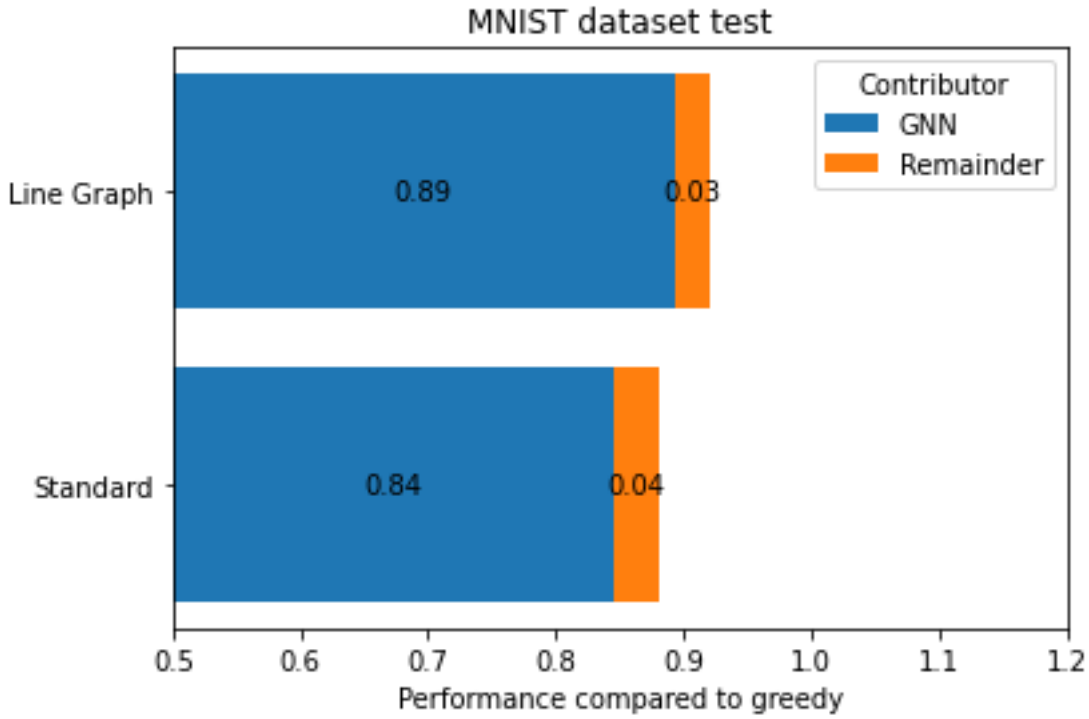

Figure 4.2: MNIST performance comparison

2. Trying to add a reduction rule and 2 additional features: 1st largest weight, 2nd largest weight for each node.

3. No noticable effect from neither reduction nor additional features.

4. Trying to train graph on the whole dataset of 55000 graphs:

RESULTS ON MNIST:

99% of greedy, remainder ¡5

RESULTS ON larger sage graphs:

sage 10: 0.86% of greedy, remainder 0.53

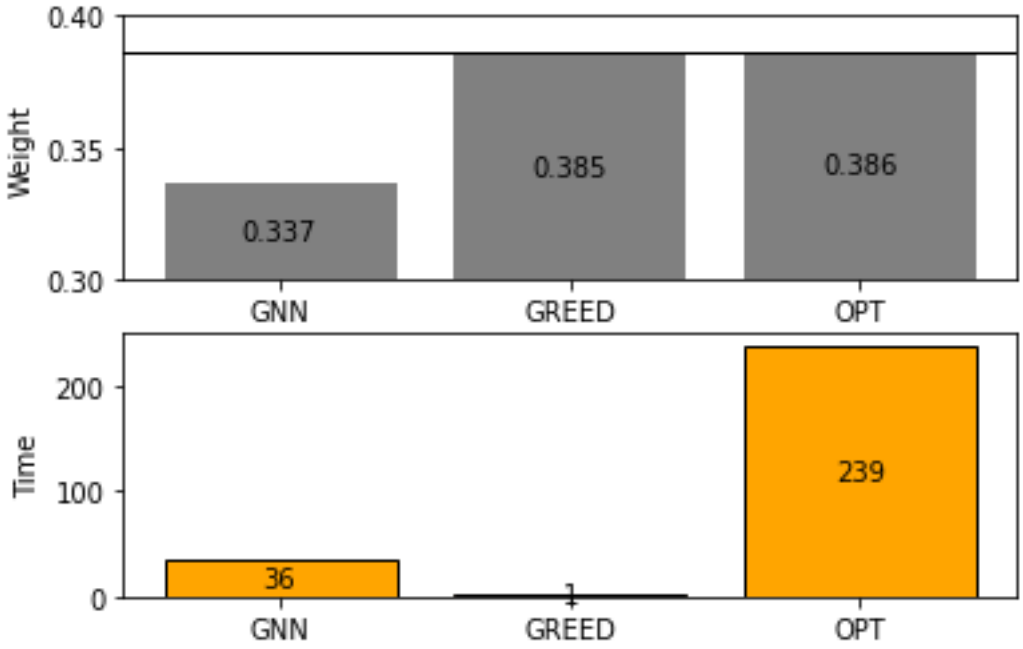

Figure 4.3: MNIST trained model performance on sage10 graph

5. Trying custom dataset of 1000+ graphs with varying sizes and structures, between 100 and 10000 nodes:

Results on MNIST:

Results on larger sage graphs:

sage 10: 94% of greedy, remainder = 0.54

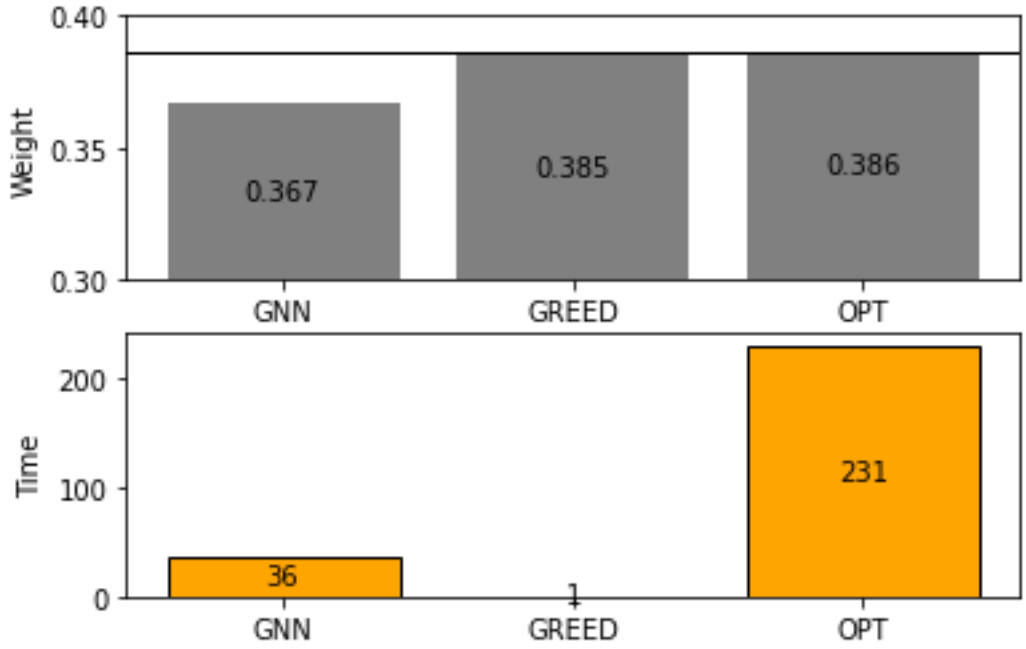

Figure 4.4: CUSTOM trained model performance on sage10 graph

6. Adjusting the threshold for the picking edges in to the solution and trying to remove edges below secondary threshold:

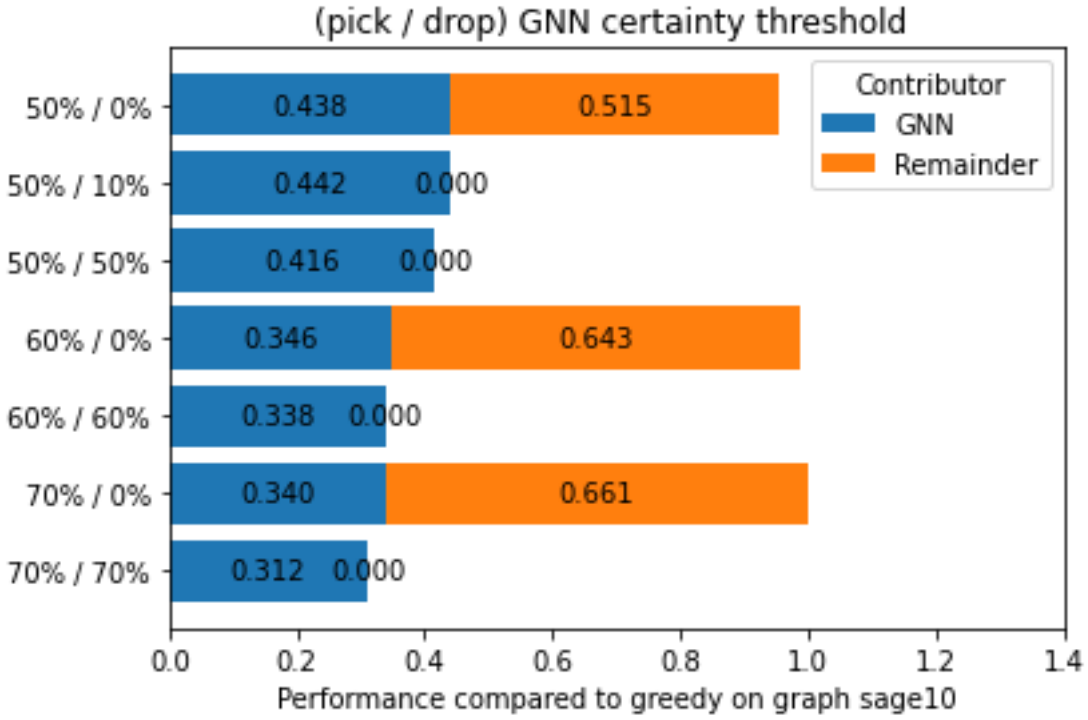

Figure 4.5: Model performance on sage10 graph

4.3 Comparison of results

4.4 Final results

Results of the best model (trained using CUSTOM dataset) on increasingly larger graphs:

4.4.1 Weakness of greedy algorithm

On average greedy seems to show realy good results, but in theory it is not hard to make a graph that abuses the greedy approach and results in poor performance. Example of a graph that GNN model easily beats:

END ——————————————————————————————- MISC NOTES IGNORE

Line graph vs normal vs normal with features (compared to standard greedy)

MNIST: 0.92 vs 0.77 vs 0.88 Normal greedy remainder LOW vs LOW vs LOW

GOOD CASE data/Pajek/GD98b/GD98b.mtx: 1.43 vs 0.00 vs 1.00 Normal greedy remainder LOW vs LOW vs LOW

MNIST ——————- GNN AVG Time = 0.1821751117706299 GREED AVG Time = 0.002601337432861328 GNN AVG Weight = 17.13489990234375 GREED AVG Weight = 17.2469970703125 remainder = 0.0440 ——————-

trained on mix MIX

MNIST with reduction ——————- GNN AVG Time = 0.2019108772277832 GREED AVG Time = 0.0028308868408203126 GNN AVG Weight = 17.117236328125 GREED AVG Weight = 16.9795166015625 weightResDif = 0.0355 ——————-

trained on mix MIX

cage10 ——————————————— with reduction trained on MNIST

TIME Opt AVG Time = 226.44823741912842 GNN AVG Time = 33.7434720993042 GREED AVG Time = 0.719895601272583 ————————- WEIGHT Opt AVG Weight = 169.5332794189453 GNN AVG Weight = 123.09953308105469 GREED AVG Weight = 169.0205456027761 ————- weightResDif = 0.32484549283981323 ———— END

TIME Opt AVG Time = 238.8848557472229 GNN AVG Time = 35.6191520690918 GREED AVG Time = 0.8406157493591309 ————————- WEIGHT Opt AVG Weight = 0.3864327073097229 GNN AVG Weight = 0.3369670510292053 GREED AVG Weight = 0.38539551471512823 ————- weightResDif = 0.5323471426963806 ———— END

cage10 ———————————————— without reduction trained on MNIST

TIME Opt AVG Time = 229.30790853500366 GNN AVG Time = 62.8546507358551 GREED AVG Time = 1.4378819465637207 ————————- WEIGHT Opt AVG Weight = 169.5332794189453 GNN AVG Weight = 90.18408966064453 GREED AVG Weight = 169.06105741672218 ————- weightResDif = 0.5952541828155518 ———— END

TIME Opt AVG Time = 228.7730164527893 GNN AVG Time = 54.679471492767334 GREED AVG Time = 1.3367598056793213 ————————- WEIGHT Opt AVG Weight = 0.3864327073097229 GNN AVG Weight = 0.2927633821964264 GREED AVG Weight = 0.38536409468088095 ————- weightResDif = 0.8586757183074951 ———— END

cage10 ———————————————— with reduction, trained on MIX

TIME Opt AVG Time = 226.55958795547485 GNN AVG Time = 38.985610246658325 GREED AVG Time = 0.7930364608764648 ————————- WEIGHT Opt AVG Weight = 169.5332794189453 GNN AVG Weight = 133.77395629882812 GREED AVG Weight = 169.0205456027761 ————- weightResDif = 0.38094136118888855 ———— END

TIME Opt AVG Time = 230.7519097328186 GNN AVG Time = 36.04787516593933 GREED AVG Time = 0.709648847579956 ————————- WEIGHT Opt AVG Weight = 0.3864327073097229 GNN AVG Weight = 0.36708855628967285 GREED AVG Weight = 0.38539551471512823 ————- weightResDif = 0.5425851345062256 ———— END

cage10 ———————————————— without reduction, trained on MIX

TIME Opt AVG Time = 237.57963299751282 GNN AVG Time = 63.44921660423279 GREED AVG Time = 1.4237451553344727 ————————- WEIGHT Opt AVG Weight = 169.5332794189453 GNN AVG Weight = 116.88045501708984 GREED AVG Weight = 169.06105741672218 ————- weightResDif = 0.7212029695510864 ———— END

TIME Opt AVG Time = 220.3951222896576 GNN AVG Time = 59.582497119903564 GREED AVG Time = 1.435647964477539 ————————- WEIGHT Opt AVG Weight = 0.3864327073097229 GNN AVG Weight = 0.35216498374938965 GREED AVG Weight = 0.38536409468088095 ————- weightResDif = 0.837394654750824 ———— END

————————————————-

cage11

without reduction

TIME Opt AVG Time = 1624.2668080329895 GNN AVG Time = 90.85399174690247 GREED AVG Time = 2.7126402854919434 ————————- WEIGHT Opt AVG Weight = 793.24462890625 GNN AVG Weight = 595.8353271484375 GREED AVG Weight = 790.162699168548 ————- weightResDif = 0.39534446597099304 ———— END

NOT line NN graph MNIST weights normalized vs MNIST weights ones vs MNIST weights 1-2 vs MNIST weights 1-5 0.88 vs 1.00 vs 0.83 vs 0.60

4.5 Performance on unseen data

Algorithm	Time	Weight
GNN	1	6
Greedy	2	7

How well the final model solves the matching problem

Chapter 5

Conclusion

Results show that GNNs are capable of solving MWM, but the approaches presented in this work showed worse performance overall comapared to a simple standard greedy algorithm. The margin between the results was not big enough to indicate that the approach was completely sensless. Compared to the greedy algorithm model showed some level of "understanding" of the task at hand and in some special cases even manages to beat the greedy algortihm. It is worth mentioning that theese cases were manualy chosen because of they abuse the naiveness of the greedy algorithm, but it does still indicate that such cases do exist and therefore there is value in using GNN instead of a greedy algorithm.

5.1 Future work

The fact that GNN in this work underperformed does not neccessary mean that the GNNs are in general unfit for MWM problem. There several potential improvements at hand. A deeper or wider model can be trained, meaning adding more layers as well as neurons to each layer to potentialy improve models ability to recognise complex patterns at the cost of longer training and prediction times. However for this particular architecture adding more layers showed little to no effect. Theres also a variety of different architectures that can be tryed out. An unsupervised approach is a good potential candidate where precomputing the optimal solution would not be needed. Instead the model can try to find the best solution by incentivising it to get as high weight sum as possible, in a way resembeling a game.

5.2 Final words

Glossary

Artificial Intelligence Artificial Intelligence is a field of study regarding intelligence simulated by computers. Where intelligence is meant in context of human intelligence.

Git Git and GitHub is a Version Control System (VCS) for tracking changes in computer files and coordinating work on those files among multiple people.

Machine Learning Machine Learning studies algorithms that learn general patterns and make predictions based on some form of input. It is one form of Artificial Intelligence.

Neural Network Neural Network is one of many technologies used inMachine Learning.

List of Acronyms and Abbreviations

CO Combinatorial Optimization.

GNN Graph Neural Network.

MIS Maximal Indepndent Set.

MWIS Maximum Weighted Indepndent Set.

MWM Maximum Weighted Matching.

VCS Version Control System.

Bibliography

[1] Maria Chiara Angelini and Federico Ricci-Tersenghi. Modern graph neural networks do worse than classical greedy algorithms in solving combinatorial optimization problems like maximum independent set. *Nature Machine Intelligence*, 5(1):29–31, December 2022. ISSN 2522-5839. doi: 10.1038/s42256-022-00589-y.
URL: `http://dx.doi.org/10.1038/s42256-022-00589-y`.

[2] Lorenzo Brusca, Lars C. P. M. Quaedvlieg, Stratis Skoulakis, Grigorios G Chrysos, and Volkan Cevher. Maximum independent set: Self-training through dynamic programming, 2023.

[3] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks, 2022.

[4] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17: 449–467, 1965. doi: 10.4153/CJM-1965-045-4.

[5] Ron Karjian. Weighted maximum matching in general graphs, 2023.
URL: `https://www.techtarget.com/whatis/A-Timeline-of-Machine-Learning-History`.

[6] Amirhossein Nouranizadeh, Mohammadjavad Matinkia, Mohammad Rahmati, and Reza Safabakhsh. Maximum entropy weighted independent set pooling for graph neural networks. *CoRR*, abs/2107.01410, 2021.
URL: `https://arxiv.org/abs/2107.01410`.

[7] Martin J. A. Schuetz, J. Kyle Brubaker, and Helmut G. Katzgraber. Combinatorial optimization with physics-inspired graph neural networks. *Nature Machine Intelligence*, 4(4):367–377, April 2022. ISSN 2522-5839. doi: 10.1038/s42256-022-00468-6.
URL: `http://dx.doi.org/10.1038/s42256-022-00468-6`.

[8] Joris van Rantwijk. Weighted maximum matching in general graphs, 2008.
URL: `https://github.com/ageneau/blossom/blob/master/python/mwmatching.py`.

[9] Bohao Wu and Lingli Li. Solving maximum weighted matching on large graphs with deep reinforcement learning. *Information Sciences*, 614:400–415, 2022. ISSN 0020-0255. doi: https://doi.org/10.1016/j.ins.2022.10.021.
URL: https://www.sciencedirect.com/science/article/pii/S0020025522011410.

Appendix A

Generated code from Protocol buffers

Listing A.1: Source code of something

```
1 System.out.println("Hello Mars");
```