Vrije Universiteit Amsterdam

Master Thesis

Leveraging Graph Neural Networks for Quadratic Unconstrained Binary Optimization Problems

Author: Olga Sergeevna Krylova (2637567)

1st supervisor:	Prof. dr. Sandjai Bhulai
1st company supervisor:	Prof. dr. Frank Phillipson
2nd company supervisor:	Dr.-Ing Esteban Aguilera
2nd reader:	Dr. Joost Berkhout

March 11, 2024

"The important thing is not to stop questioning. Curiosity has its own reason for existing."

Albert Einstein

Abstract

Recent advances in deep learning techniques pose a question of whether they can facilitate the task of finding good quality solutions to combinatorial optimisation (CO) problems in a practically relevant solution time. Specifically, it is of practical relevance to determine to which extent graph neural networks (GNNs) can be applied to CO problems that can be formulated as QUBO's and thus be naturally interpreted as graph problems. In this research a GraphSAGE based GNNs combined with Rprop optimizer are applied to the two classical CO problems - the maximum cut problem and maximum independent set problem - in an unsupervised learning setting to measure their potential and limitations imposed by the graph density and discrete nature of the tasks at hand. The GNN-solver performance is further evaluated on portfolio selection case performed on Nikkei225 stock index. It is demonstrated that GNN solver is capable to derive portfolios that are very close to Pareto front when provided with a suitable loss function.

Acknowledgements

I would like to express my deepest gratitude to both of my primary supervisors, Professor Sandjai Bhulai and Professor Frank Phillipson, who guided me throughout this project with their meaningful questions and insightful remarks, always respecting my autonomy and independence in this research. I deeply appreciate the freedom I was given in choosing my experimental path through the topic and constant support and encouragement to go further.

I would also like to thank my second supervisor form the TNO side dr. Esteban Aguilera for sharing his ideas and tips that enriched my investigation of the topic and helped me in the right direction.

Next, I would like to extend special thanks to Professor Rob van der Mei and to Annemieke van Goor, the Business Analytics internship coordinator, for helping me to find such an interesting and engaging internship topic by bringing me in contact with Frank.

Lastly, I would like to thank my family for their patience and support and specifically my husband, Igor, who has spent a lot of summer weekends and evenings listening to me thinking out loud about all things GNN, QUBO and gradient descent related without any complaints and with unwavering trust in my ability to successfully complete this topic.

Contents

List of Figures **vii**

List of Tables **ix**

1 Introduction **1**

2 Related Work **4**

3 Background **9**

3.1 Quadratic Unconstrained Binary Optimization (QUBO) 9

3.2 Methods 10

3.2.1 Gradient Descent 10

3.2.2 Graph Neural Networks 10

3.2.2.1 Graph Convolutional Networks 12

3.2.2.2 Graph Sage 12

3.2.2.3 Optimizers 13

4 Numerical Experiments Design **16**

4.1 Problems 16

4.1.1 Maximum Cut Problem (MC) 16

4.1.2 Maximum Independent Set Problem (MIS) 17

4.1.3 Instances 18

4.2 Methods 19

4.2.1 Baselines 19

4.2.1.1 Heuristic for the Maximum Cut Problem 19

4.2.1.2 Heuristic for the Maximum Independent Set Problem . . . 19

4.2.1.3 Gradient Descent 20

4.2.1.4 Analytical solution to the continuous version of $\mathbf{x}^T \boldsymbol{Q} \mathbf{x}$ . . . 21

4.2.2 Graph Neural Network solver 22

4.3 Experimental setup and hyperparameters 25

5 Numerical Experiments Results **27**

5.1 Gradient descent methods 27

5.2 Graph Neural Network solvers 30

5.2.1 Max Cut 30

5.2.2 MIS 31

6 Portfolio optimization using GNN-solver **38**

6.1 Portfolio Optimization 38

6.1.1 Problem description 38

6.1.2 QUBO formulation 39

6.2 Method 40

6.3 Experimental Setup 42

6.4 Data 42

6.5 Results 44

6.5.1 Experimental Setup 1 44

6.5.2 Experimental Setup 2 45

6.5.3 Experimental Setup 3 49

6.5.4 Experimental Setup 4 53

7 Conclusion and discussion **61**

References **64**

List of Figures

3.1 Message passing architecture. Toy example. 11

4.1 Diagram of GNN architecture. 23

5.1 Comparison performance of plain gradient descent on Max Cut and MIS problems. Regular graphs. Left: Average fraction of "classical" solution found. Right: Average fraction of classical algorithm runtime spent. 28

5.2 Comparison performance of plain gradient descent on regular and probabilistic graphs. 29

5.3 Performance per gradient descent method. Regular graphs. 29

5.4 Average fraction of "classical" solution found by the basic GNN solver. . . . 31

5.5 Max Cut. Regular graphs. Comparison of the baseline GCN-solver with the Adam vs. Rprop optimizer. 32

5.6 Average size of the independent set found by the basic GNN solver on regular graphs. 32

5.7 Comparison of the Adam vs. Rprop optimizer. GCN-solver. Regular graphs. Left: Average fraction of "classical" solution found. Right: Average runtime. 33

5.8 Difference in performance per graph type for GNN solver with Adam and with Rprop. 33

5.9 GNN-solver with embedding transfer. Regular graphs 34

5.10 GNN-solver with and without embedding transfer. Regular graphs. Left: Average fraction of "classical" solution found. Right: Average runtime. . . . 35

5.11 Comparison of the GNN-solver with frozen and not frozen embeddings transfer. Regular graphs. 35

5.12 GraphSAGE-solver vs. GCN-solver with embedding transfer 37

6.1 Nikkei225 assets. Left: Original data Right: Transformed data. 43
6.2 Histogram of Nikkei225 assets original returns. 44
6.3 Returns and risks of 500 randomly selected portfolios of sizes 20 and 50 compared to minimal required return R. 44
6.4 Histograms of portfolio sizes selected by GNN solver per experiment parameters. $N = 225$ 45
6.5 Portfolios selected by GNNSolver compared to random portfolios of the same size. $N = 225$, $n = 20$, $R = 0$ 46
6.6 Portfolios selected by GNNSolver compared to random portfolios of the same size. $N = 225$, $n = 20$, $R = 3500$ 46
6.7 Average and best portfolios per hyperparameter combination compared to 100 randomly selected portfolios of comparable sizes. The dot sizes of the solved portfolios correspond to their respective sizes. Random portfolios are plotted without sizes. 48
6.8 Average statistic per hyperparameter group. 49
6.9 Portfolios selected by the GNNSolver compared to random portfolios of the same size. $N = 100$. 50
6.10 Portfolios selected by the GNNSolver compared to random portfolios of the same size. $N = 225$ 50
6.11 Histogram of GNNsolver selected portfolios sizes per N. 51
6.12 Use of Nikkei225 assets for portfolio selection by the GNN-solver. 52
6.13 Comparative selection frequency of Nikkei225 assets per N. 53
6.14 Histogram of respective portfolio sizes found per weight on returns. 54
6.15 Pareto fronts: random, Gurobi, and best 5 per weight selected by GNNsolver. 56
6.16 Mean GNN-solver solution time per λ and N. 57
6.17 "Portfolios selected by GNN and Gurobi compared to random portfolios per portfolio size. $N = 150$" 57
6.18 "Portfolios selected by GNN and Gurobi compared to random portfolios per portfolio size. $N = 225$" 58
6.19 Pareto front per number of runs of GNN-solver per instance/lambda combination compared to Gurobi Pareto front. 59
6.20 Solution time per point of Pareto front. 60

List of Tables

4.1 List of numerical experiments. 24
4.2 Hyperparameters per gradient descent setup. 26

1 Introduction

Many of the problems that individuals and businesses face on a day-to-day basis are *optimization problems* which deal with selecting the best solution to a particular situation. Optimality is a natural prerequisite for the efficiency of businesses or services provided, e.g., cost minimization or revenue maximization. Historically, this efficiency was primarily facilitated through *Operations Research* - a scientific approach that helps decision makers through application of quantitative methods (1). This field of study started in the Second World War "as an initiative to use mathematics and computer science to assist military planners in their decisions" (2). Presently, it covers such industries as transportation, financial and banking sectors, supply chains, electrical and traffic networks.

Depending on the nature of the decision variables involved, optimization problems can be continuous or *discrete.* The subset of the discrete optimization problems is formed by *combinatorial optimization* (CO) problems, that are dealing with discrete solution spaces, that usually consist of permutations, subsets or orderings of the discreet decision variables.

Significant number of the CO problems are known to be difficult. In fact, in terms of computational complexity theory, many of them are known to be *NP-hard* (3). NP-hardness boils down to the fact that finding optimal solutions to the real-life sized problems is often not feasible within practically relevant solution time. However, since day-to-day existence and activities are often reliant on the solutions to such problems, they are being solved in practice by heuristics and approximate algorithms. Neither of both are guaranteed to deliver an optimal solution, however, in practice having a solution of a reasonable quality within a feasible solution time is enough. Nevertheless, categories such as "reasonable quality" and "feasible solution time" typically offer room for improvement and thus more and more scientific disciplines get involved with the field of CO. In recent years, first machine learning and then deep learning approaches specifically started to being applied to the CO problems (for comprehensive review of use of ML approaches for CO problems see e.g., (4), (3)).

It must be noted that a lot of CO problems are graph-related. This relation can exist in three possible ways. First, a CO problem can be directly defined on a graph. Examples of such problems are the *maximal clique* problem or *the minimal vertex cover* problem. Next, some problems although not formulated on graphs can also be formulated as graph problems. An example of such a problem is the *set covering problem.* Lastly and most importantly, some CO problems irrespective of their type can be viewed as a graph, where variables represent graph vertices and interactions between those variables represent edges. This point is being elaborated in Section 3.1. Alternatively, "the interaction between variables and constraints in CO problems naturally induces a bipartite graph, i.e., a variable and constraint share an edge if the variable appears with a non-zero coefficient in the constraint" (5). To summarize the above – graph structures arise naturally in CO, however, for a long time graphs as an unstructured data posed a challenge for conventional machine learning methods and neural networks specifically. The emergence of *graph neural networks* (GNN) architectures that are capable of working directly with graphs as input stimulated a wave of new approaches and applications of deep learning to CO problems.

Kotary et al. (3) defines two main ways in which GNNs can enhance the solution processes. GNNs can be used to assist existing solvers in the solution process, forming so-called ML-augmented CO. These methods "learn to guide the search decisions in branch and bound solvers, and methods that guide the application of primal heuristics within branch and bound" (3). On the other hand, GNNs can be used for end-to-end learning as a function that maps a particular problem instance to a solution.

From the machine learning perspective, three main approaches exist: supervised learning, unsupervised learning, and reinforcement learning. Supervised learning with GNNs with respect to CO problems poses a considerable challenge at the moment, since it requires a set of solved CO instances, which is difficult to obtain due to the NP-hardness. Nevertheless, there is a considerable amount of research into the topic (see, e.g., (6), (7), (8)). The reinforcement learning paradigm is a very promising research direction for CO problems since on the one hand it does not require a set of already solved NP-hard problems and on the other hand it can mimic existing algorithms and heuristics that often construct a solution iteratively. Examples of using GNNs within the reinforcement learning paradigm with respect to CO problems are the works of Dai et al. (9) and Kool et al. (10). Finally, unsupervised learning potentially provides the simplest and the most forward approach to solving CO problems with GNNs, which does not require presolved training set of problems and uses provided cost function as a loss to minimize in the process of a network training.

Current research was broadly inspired by the work of Schuetz et al. (11), where GNNs were applied to two CO problems formulated as *Quadratic Unconstrained Binary Optimization Problems* in an unsupervised manner. The unconstrained nature of QUBO formulations offers easily utilizable loss functions for deep learning. Although binary decision variables lead to a non-continuous loss functions, a straightforward relaxation to the continuous variable $p \in [0, 1]$ that can be interpreted as a probability of a binary variable to be equal to one, was repeatedly proposed in the previous research. According to Schuetz et al., "the graph neural network optimizer performs on par or outperforms existing solvers, with the ability to scale beyond the state of the art to problems with millions of variables." They also list a few real-life applications for the GNN-solver. While demonstrating very promising results, this work is somewhat constrained in scope. Namely, the GNNs in both cases were applied to very sparse graphs only without any indication of how their performance is influenced by the increase in connectivity of the problem graph. Another limitation of the study is that while it mentions potential real-life applications, no evidence is given to the quality of the results achieved by the GNN-solver on a real-life case.

In this work, the application of the GNN-solver to QUBO problems is studied in further detail to answer the question to which extend Graph Neural Networks can be leveraged to solve QUBO's. Specifically, it is important to determine which features of the QUBO at hand might impair the GNN-solver performance; which GNN architectures and to which extent can assist in overcoming these obstacles; and which difficulties might be considered inherent to the GNN-solver. To this end experiments conducted in (11) are extended in this work to higher degree graphs to investigate the influence of increasing graph connectivity on the GNN-solver performance. Also performance of the GNN-solver is evaluated on the real-life use-case, namely, the *portfolio optimization problem* in its QUBO formulation. The performance is discussed in terms of the quality of the solutions found compared to the random selection and its respective solution time.

This paper is structured as follows. Section 2 provides more insight into the recent advances in the topic that were already briefly outlined above. In Section 3 some preliminary concepts are discussed in more details such as definition of the QUBO and Graph Neural Network architectures. Sections 4 and 5 are dedicated to the description of the numerical experiments that were conducted and their results. Section 6 presents a portfolio optimization case with the GNN-solver. Finally, Section 7 summarizes the results, discusses limitations of the study and offers avenues for the potential further research.

2

Related Work

A comprehensive survey of machine learning methods applied to CO problems that can be defined on graphs was published by Vesselinova et al. in 2020 (4). Apart from covering recent research on the topic of applying machine learning to CO problems, the authors also classify proposed methods by the machine learning method/architecture (attention, transformer, GNN, etc.) applied and more generally by the machine learning paradigm used (supervised vs. reinforcement learning). The study, however, does not mention any research where GNNs were applied to the graph problems in an unsupervised learning setting.

Another general overview of the recent applications of machine learning methods to combinatorial optimization problems is presented in the survey of Kotary et al. (3). The scientific papers discussed there are also divided into the supervised learning and reinforcement learning methods. Furthermore, the authors also provide an extended overview of what they call the "Predict-and-Optimize paradigm" within the "End-to-End CO learning" (E2E-COL), which focuses on integrating combinatorial solvers or optimization methods into deep learning architectures". The main defining feature of this family of methods according to the authors, is that they use cost function of a particular optimization problem as a loss function for a corresponding machine learning task. Since the main machine learning tool - backpropagation - is fully dependent on gradients of the loss function, the survey rightfully points out and discusses the problem that "combinatorial problems with discrete state spaces do not offer useful gradients". This problem was investigated in the researches of Donti et al. (12), Amos and Kolter (13), and Mandi and Guns (14) to name a few.

Specifically, Vlastelica et al. (15) state, that "the fundamental problem with differentiating through combinatorial solvers is not the lack of differentiability; the gradient exists almost everywhere. However, this gradient is a constant zero and, as such, is unhelpful for optimization". They propose a way to backpropagate the linear loss through a blackbox combinatorial solver. In particular their approach is to learn a meaningful representation for a blackbox solver of the problem input w through the neural network convolutional layers. The solver then maps this continuous input w to the discrete solver output $y(w)$, which minimizes the given linear loss function. They propose a way of computing the loss gradient with respect to the solver input. The use of convolutions to derive meaningful problem representations together with the problem of backpropagation with discrete loss function is what relates their paper to the current research.

An extensive overview of the GNN applications to combinatorial tasks is offered in a recent survey of Cappart et al. (5). The authors classify again all reviewed approaches by the used machine learning paradigm (supervised, unsupervised and reinforcement learning), but also classify the existing approaches by the general type of task at hand. Specifically, they differentiate over use-cases where GNNs are used to produce a solution to the CO problem, use-cases where the task is to prove the optimality of the solution potentially produced by another solver or investigate the optimality gap and combining GNNs with other CO algorithms for neural algorithmic reasoning.

The paper also lists a number of limitations of the current GNNs and problems that those limitations pose for the CO tasks. One of the limitations of the GNNs is their limited expressivity or representation ability studied by Morris et al. (16) and Garg et al. (17). It was demonstrated that graph representations produced by GNNs might not be discriminating enough to guide the solution process. This problem is dependent among others on the maximal degree of the graph (5). Another limitation is referred as the bottleneck problem, which arises form the limited ability of the GNNs to capture and retain global graph and long-range neighborhood information.

The unsupervised learning paradigm was realized Within the current research. As was mentioned before, unsupervised learning directly utilizes the CO problem cost function to arrive to the solution, thus no solved problem instances are required. A more detailed review of the recent research over GNN applications to the CO problems in an unsupervised way is given bellow.

Karalias and Loucas (18) described a learning framework that can be applied for the wide range of CO problems. Within the proposed framework, first the GNN is used to fit the probability distribution over the nodes, where each node is being assigned a probability of belonging to the optimal solution. In the next step, an optimal solution is being recovered either by sampling or by the method of conditional expectation. In the second case, the solution is constructed sequentially, which prevents constraint violations and ensures feasibility. The authors demonstrate that this approach outperforms many existing deep learning algorithms and classical solvers in terms of the quality of the solution and scalability.

Lemos et al. (7) applied the GNNs to the decision version of the graph coloring problem. Their approach is to extend the actual graph that needs to be colored with the extra nodes per color available (in total C colors). The color nodes are fully connected. GCN is then applied to the resulting graph to produce embeddings for the graph nodes that are further being fed to the MLP. The MLP is assigned a binary classification task, namely, it has to decide if the graph at hand accepts C-coloration. To find a chromatic number for a specific graph, this graph is then fed to the trained network with a decision question for a range of C values. The authors report that the accuracy achieved by the proposed method was lower than the accuracy of the Tabucol and greedy algorithm that they have used. The authors of the paper also investigated the quality of the embeddings produced by GCN and their discriminating power. They clustered produced embeddings into C groups with a straightforward assumption that node embeddings should correspond to the color that should be assigned to them within C-coloring scheme. Then they measured a number of coloring conflicts within the cluster and reported that the number of conflicts was growing with the chromatic number. It should be noted on the side that the chromatic number is higher for the graphs with higher connectivity.

Schuetz et al. (2022b) (19) also applied GCN to the graph coloring problem. In their approach, the authors defined the decision variables as a vector $\mathbf{y}^j$ per node j of length q, where q is the number of colors to be used for graph coloring with the constraint that $y_i^j \in \{0, 1\}$ and $\sum_{i=1}^{q} y_i^j = 1$. To produce a differentiable loss function, they applied a relaxation to the $y_i^j \in 0, 1$ and instead used $p_i^j \in [0, 1]$, which can be interpreted as a probability of color i to be assigned to the vertex j. They have further defined an unconstrained loss function inspired by the Hamiltonian of the Potts model. The authors proposed and compared two GNN architectures: GCN and GraphSAGE. The solutions derived for a set of a benchmark graphs demonstrated superior performance of GraphSAGE based architecture, that performed on par with Tabucol algorithm.

Yao, Bandeira and Villar (20) applied a GNN to the maximum cut (Max-Cut) problem, which can be formulated as a binary quadratic unconstrained optimization problem. They evaluated their approach on random regular graphs with low connectivity. The authors report that GNNs perform on par with other existing methods.

Xu, Hui and Fu in their paper from 2020 (21) apply a variation of the GNN architecture named TilinGNN to the 2D tiling problem. For the problem to be tackled by the proposed GNN solver, they define it on a graph, where the nodes are potential locations for a tile of a specific shape to be placed and two types of edges between nodes are introduced - overlapping edge and neighbouring edge. They further create an unconstrained loss function that imposes a positive loss per violated constraint. The decision variable $x_i \in [0, 1]$ that is being output by the GNN is defined as probability of a tile to be placed at location i. The authors show that their approach works for several tiling sets with a linear scaling of solution time to the number of candidate placements for tiles. As the limitations of their work, they mentioned the inability to work with specific type of inputs such as Penrose tile sets, and inability to impose hard constraints.

The current research was inspired by a simple and straightforward unsupervised approach, suggested by Schuetz et al. (2022a) (11).The authors applied a GCN to several CO problems formulated as QUBOs in an unsupervised manner, using the QUBO cost function as a loss to be minimized during the "training" process. However, there is no actual training happening, since the GCN is applied to a specific QUBO and is not expected to generalize to other problems. In fact, the GCN is used to produce a candidate solutions iteratively with the idea that the solution is improved at each iteration by GCN. In this respect, the research is close to the framework that was suggested by Karalis and Loucas, with the blackbox solver being replaced by a simple sigmoid function. As was mentioned above, the QUBO formulation works with binary variables that pose a challenge for deep neural networks, requiring continuous loss function. The usual approach that was also used by the authors in the described paper is to apply a relaxation and work with a continuous variable $p_i \in [0, 1]$ that can be interpreted as $p(x_i = 1)$. In the research, this unsupervised approach was applied to the Max-Cut and the maximal independent set problems defined on a series of generated graphs. The authors compared their solutions for the Max-Cut problem to the solutions derived with the Goemans-Williamson approximate algorithm and analytically derived upper bounds. The reported cut sizes found by the GCN-solver were close to the theoretical upper bound. The GCN-solver performed similarly well on the MIS instances. According to the paper the solution time for the GCN-solver scales linearly with number of nodes n. Another benefit of the GCN-solver is that the quality of the reported solutions does not decrease as n grows. However, all the experiments were performed on the graphs with degree 3 and 5, which can be considered very sparse especially as the number of nodes grows. This leaves a lot of questions about the performance of the GCN-solver on denser graphs especially in view of the GCN limitations discussed in the researches mentioned above. The authors also briefly discuss potential industry applications of the GCN-solver, however, no actual real-life case results are reported.

The current research is aiming to overcome these limitation by extending the experiments discussed above to the wider set of graphs with higher densities, by comparing other GNN architectures and their impact on the GNN-solver's performance, and by evaluating the GNN-solver performance on the real-life case of the optimal portfolio selection.

3

Background

3.1 Quadratic Unconstrained Binary Optimization (QUBO)

The quadratic unconstrained binary optimization problem (QUBO) was first studied in the middle of the last century as a subclass of pseudo-Boolean optimization problems. Lately, the academic interest in research topics regarding QUBO has peaked again, spurred by the advances in quantum computers and quantum annealers. Abraham Punnen in (22) gives the following definition of QUBO. "Let $\boldsymbol{Q}$ be an $n \times n$ matrix, $\mathbf{c}^T = (c_1, c_2, \ldots, c_n)$ be a row vector from $\mathbb{R}^n$, and $\mathbf{x}^T = (x_1, x_2, \ldots, x_n)$ be a row vector from $\{0,1\}^n$. Then, QUBO is defined as the mathematical programming problem:

$$\text{Maximize} \quad \mathbf{x}^T \boldsymbol{Q} \mathbf{x} + \mathbf{c}^T \mathbf{x} \tag{3.1a}$$

$$\text{Subject to} \quad \mathbf{x} \in \{0,1\}^n \tag{3.1b}$$

".

The linear term can be incorporated into the $\boldsymbol{Q}$ matrix since for the binary variable y the following is true: $y^2 = y$. To get the adjusted $\tilde{\boldsymbol{Q}}$, all diagonal entries should be increased by respective c, specifically: $\tilde{q}_{ii} = q_{ii} + c_i$ for $i = 1, 2, \ldots, n$

Punnen in (22) also gives a graph interpretation of QUBO that is especially relevant to this work and is cited here for completeness. A graph $\mathbf{G}$ with vertex set $\mathbf{V}$, where $\mathbf{V} = \{1, 2, \ldots, n\}$, and edge set $\mathbf{E}$, where edge $(i,j) \in \mathbf{E}$ if $q_{ij} \neq 0$, is called the support graph of $\tilde{\boldsymbol{Q}}$. The diagonal entries of $\tilde{\boldsymbol{Q}}$ represent costs of the vertices and off-diagonal costs of the edges. Then the QUBO seeks a subset $\mathbf{S}$ of $\mathbf{V}$ such that the cost of the induced subgraph $\mathbf{G}(\mathbf{S})$ is maximized, where the cost of a subgraph is the total cost of all included vertices and edges. For example, when solving a Maximum Clique problem, one might be interested in finding such a subset of variables that are all adjacent to each other.

In the current work, a minimization variant of QUBO is treated, which can be trivially derived from the version described above by taking $\boldsymbol{Q} = -\tilde{\boldsymbol{Q}}$.

3.2 Methods

3.2.1 Gradient Descent

Gradient descent is a first-order optimization algorithm that can be iteratively applied to solve unconstrained optimization problems of the form

$$\min f(x), \quad x \in \mathbb{R}^n, \tag{3.2}$$

where cost-function $f : \mathbb{R}^n \to \mathbb{R}$ is uniformly convex and twice continuously differentiable. The iterative process, as described by Curry in (23) has the following form :

$$x^{k+1} = x^k + tz^k, \tag{3.3}$$

where t is a positive step-length and $z^k = -\frac{\partial f(x)}{\partial x^k}$ is a cost-function gradient with respect to the current solution x_k.

It is a method for local optimization in the sense that it searches for the local minima, but it is not a local search algorithm, because it does not perform an explicit search of the local neighborhood in the solution space.

Curry suggested to choose the optimal step-size by trial and error. In the recent decades, due to the wide use of gradient descent for training of neural networks, a lot of effort has been put into developing effective schemes of determining optimal (often time-dependent) step sizes. They are considered below in Section 3.2.2.3.

3.2.2 Graph Neural Networks

Graph Neural Networks are one of the relatively recent additions to the family of deep neural network architectures. The applications include but are not limited to physics, chemistry, biology, and linguistics (24). Following successful applications of deep neural networks on Euclidian data, graph neural networks extend those approaches to non-Euclidian datasets. GNN is now an umbrella term that incorporates a variety of different neural networks that are able to take graphs as their input and usually operate through so-called message-passing (25). The message-passing procedure in graph neural networks is a way to aggregate graph information from some neighborhood of a graph node to be

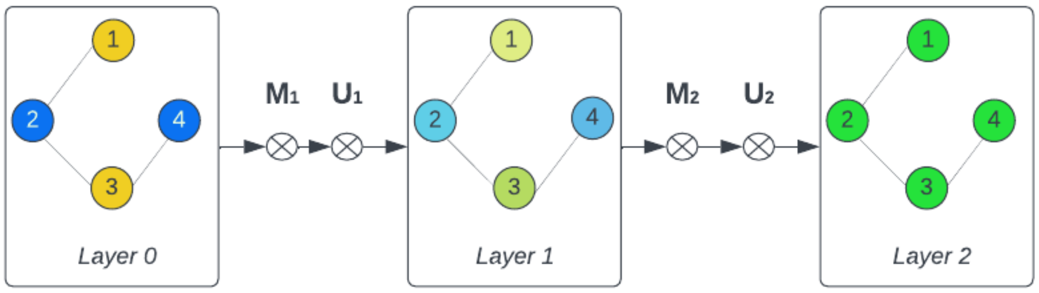


Figure 3.1: Message passing architecture. Toy example.

incorporated in the representation of this node in the next layer of the network. Each layer t represents an aggregation step. Mathematically, this can be expressed as (26):

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{v,w}), \tag{3.4}$$

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1}), \tag{3.5}$$

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

Figure 3.1 shows a simplified example of a message-passing procedure on graphs. In this example, the input is a graph, where each node is represented by a value $x_i \in [0, 1]$. The representation of the graph at level 0 ($t = 0$) is $\mathbf{h}_0 = [0, 1, 0, 1]$, where 0 corresponds to the yellow color in the scheme and 1 to the blue. M_t in this example is chosen to be a simple neighborhood average, and U_t is a weighted average between $\mathbf{m}^1$ and $\mathbf{h}^0$ with respective weights $[0.25, 0.75]$ at $t = 0$ and $[0.5, 0.5]$ at $t = 1$. Then $\mathbf{m}^1 = [1, 0, 1, 0]$, which can be interpreted as an average color of node neighbors and $\mathbf{h}^1 = [0.25, 0.75, 0.25, 0.75]$ as a weighted average representation of the nodes color and the color of its neighborhood. After the second aggregation step, $\mathbf{m}^2 = [0.75, 0.25, 0.75, 0.25]$ and $\mathbf{h}^2 = [0.5, 0.5, 0.5, 0.5]$. This example also shows why graph neural networks are usually shallow - deep architectures lead to homogeneous representations of all nodes.

Among many graph neural network architectures that currently exist, this research deals with two: Graph Convolutional Networks and SageGCN.

3.2.2.1 Graph Convolutional Networks

The Graph Convolutional Network is probably one of the most popular GNN architectures. It was introduced in 2017 by Thomas Kipf and Max Welling (27). As its name suggests, GCN performs a convolution/aggregation of node information on some neighborhood. Given the graph **G** with the set of nodes **V**, the set of edges **E** and adjacency matrix A, the GCN layer can be expressed as follows:

$$H^{t+1} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^tW^t), \tag{3.6}$$

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

This operation (3.6) is applied several times such that the representation of a graph after it can be seen as the next layer of GNN. GNNs are usually shallow networks, since after performing too many convolution steps, neighborhood information propagates further and further and all nodes become homogeneous. This problem is well-studied and referred to as oversmoothing (28).

As was shown in (25), Equation 3.6 can be rewritten for a node i to offer more insight into the inner workings of the GNN:

$$H_i^{t+1} = \sigma(\sum_{j\in N(i)} \frac{A_{ij}}{\sqrt{\tilde{D}_{ii}\tilde{D}_{jj}}} H_j^t W^{t+1} + \frac{1}{\tilde{D}_{ii}} H_i^t W^{t+1}). \tag{3.7}$$

This rewritten form sheds light on the inner workings of the graph convolution layer. The whole operation is a combination of two simpler processes. The first part of the equation under the summation sign performs neighborhood information aggregation in this case by taking a weighted average of representations of all the nodes adjacent to v_i, and this aggregate is added to the node representation of itself obtained from the previous layer normalized by its degree. The aggregation subroutine can be further generalized to include other aggregation functions, which was done in the GraphSAGE module discussed below.

The graph representations learned by this procedure can be used for graph and individual node classification tasks, node prediction, community detection, etc.

3.2.2.2 Graph Sage

The GCN architecture described above is transductive, meaning that it generates representations for the nodes that are present at the training. Adding new nodes to the graph will require retraining. As opposed to that, an inductive architecture was suggested by Hamilton, Ying, and Leskovec in (29) - GraphSAGE. The idea behind this architecture is very similar to the GCN layer described above, with three major difference:

- From Equation (3.7) it can be observed that the GCN incorporates information about a node neighborhood into its representation by aggregation. The authors of GraphSAGE suggested concatenation of these two different pieces of information. By doing this more information is preserved, namely information about a node itself and its locality in the graph is kept separately (30).
- The GCN training happens on the whole graph, while GrapSAGE, in its minibatch version, performs training on the randomly sampled subgraphs.
- Finally, instead of using a weighted average as an aggregation function as GCN does, GraphSAGE does not impose a particular choice of the aggregate function. The authors investigate in their paper a simple mean, a LSTM (Long Short Term Memory) layer as an aggregation function and different versions of the pooling functions applied per feature in embedding, e.g., mean and max pooling.

3.2.2.3 Optimizers

The most common technique to train neural networks is by optimizing their weights with gradient descent. The plain or vanilla gradient descent, as described in 3.2.1, suffers from several drawbacks that are neatly summarized in (31). Even Curry in (23) already points out the necessity to choose the correct step size for the algorithm to converge efficiently, which is not a trivial task, since the flatter regions of the cost-function surface might require larger step-sizes than the steeper ones. Another challenge is saddle points, where the plain gradient descent can easily be trapped. Moreover, different step sizes might be required per directioin the case of the multivariate input. All these shortcomings of the plain gradient descent spurred the development of various different gradient descent optimization algorithms that aim to speed up the process of convergence. A short overview of the ones that were utilized in this research is given below.

Momentum One of the simplest methods to quickly cross through the saddle points and regions that are flat in some directions is momentum. The momentum term accumulates information from the previous updates: and gradually increases step sizes in which direction of the gradient is consistent. In physical sense it mimics the velocity applied to the moving object. The gradient evaluated at the current point is not applied directly to the solution but rather influences it indirectly through the momentum term ("velocity"). Mathematically, this adaptation of gradient descent from equation (3.11) looks as follows:

$$\begin{aligned} m^k &= \gamma m^{k-1} + t\nabla_k f(x), \\ x^k &= x^{k-1} - m^k, \end{aligned} \tag{3.8}$$

where v^k is a recursively calculated quantity, γ is a momentum term that controls the fraction of the previous update that is going to be reused (γ is commonly set to 0.9) and t is a learning rate as in plain gradient descent.

Adam Adam is one of the most popular optimizers for training neural networks, and is usually appreciated for its speed, ability to deal with sparse gradients and computational efficiency (32). The name of the algorithm refers to "adaptive moment estimation". Similar to momentum, Adam accumulates the previous gradients as their moving average:

$$m^k = \beta_1 m^{k-1} + (1-\beta_1)\nabla_k f(x). \tag{3.9}$$

It also stores the moving average of the previously squared gradients:

$$v^k = \beta_2 v^{k-1} + (1-\beta_2)(\nabla_k f(x))^2 \tag{3.10}$$

Since both measures are biased towards zero in the beginning, they are corrected:

$$\begin{aligned} \hat{m}^k &= \frac{m^k}{1-\beta_1^k}, \\ \hat{v}^k &= \frac{v^k}{1-\beta_2^k}. \end{aligned} \tag{3.11}$$

Finally, the actual update step Equation (3.11) looks as follows:

$$x^{k+1} = x^k - \frac{t}{\sqrt{\hat{v}^k} + \epsilon}\hat{m}^k. \tag{3.12}$$

β_1, β_2 and ϵ are parameters to choose. However, default values, suggested by the authors of the algorithm in (32): $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$, are usually good enough.

Rprop The last optimizer that is being reviewed here is rarely used in recent research. Rprop or resilient backpropagation method was introduced in 1992 (33). Contrary to other methods described above, the algorithm uses only the sign of the derivative and does so independently per parameter. The full description of the algorithm can be found in the paper cited above. In this research, Rprop was chosen as one of the optimizers for its considerable speedup. However, the crudeness of the update that is based only on the sign of the gradient can in some cases be detrimental to the solution process, but sometimes works beneficially. This is further discussed in Section 5.

4

Numerical Experiments Design

This section describes the general setup of the experiment: describes the problems used for benchmarking and gives an overview of the methods used to obtain solutions for the problems.

4.1 Problems

Following (11), two well-studied combinatorial optimization problems were chosen for the numerical experiments: the maximum cut problem and the maximum independent set problem.

4.1.1 Maximum Cut Problem (MC)

Given an undirected graph $\mathbf{G}$ with a set of nodes $\mathbf{V}$ and a set of edges $\mathbf{E}$, the partitioning of $\mathbf{V}$ into two complementary subsets $\mathbf{V}_1$ and $\mathbf{V}_2$ is being searched for, such that the number of edges between those subsets of nodes is maximal. This was formulated in (34) as follows.

Let $\mathbf{x} \in \{0,1\}^{|\mathbf{V}|}$, then

$$\max y = \sum_{i,j \in E} (x_i + x_j - 2x_i x_j), \tag{4.1}$$

where x_i is the state i-th variable/node $i = 1, 2, \ldots, |\mathbf{V}|$.

This problem can be cast into a minimization problem and rewritten as (11)

$$\min y = \sum_{i<j} \boldsymbol{A}_{ij}(2x_i x_j - x_i - x_j), \tag{4.2}$$

where $\boldsymbol{A}$ is an adjacency matrix of $\mathbf{G}$. Mohades and Kahaei (35) show the Maximum Cut Problem in a QUBO matrix formulation:

$$\min y = \mathbf{x}^T \boldsymbol{Q}\mathbf{x}, \tag{4.3}$$

where $\boldsymbol{Q} = \boldsymbol{A} - \boldsymbol{D}$ and $\boldsymbol{D}$ is a degree matrix of $\mathbf{G}$. MC is an NP-hard optimization problem (36). However, multiple approximation algorithms have been developed. The best-proven approximation algorithm achieves 0.87856 of the optimal value (37). We also note that gradient based algorithm for solving MC problem was described in (35).

4.1.2 Maximum Independent Set Problem (MIS)

Balakrishnan and Ranganathan in (38) give the following formulation of the Maximal Independent Set problem. Given a graph $\mathbf{G}$ with vertex set $\mathbf{V}$ and edge set $\mathbf{E}$, a subset $\boldsymbol{S}$ of $\mathbf{V}$ is independent if no two vertices of $\boldsymbol{S}$ are adjacent in $\mathbf{G}$. Definitions of maximal and maximum independent sets exist. $\boldsymbol{S}$ is a maximal independent set of $\mathbf{G}$ if it is not a proper subset of any other independent set of $\mathbf{G}$. $\boldsymbol{S}$ is a maximum independent set of $\mathbf{G}$, if $\mathbf{G}$ has no independent set $\boldsymbol{S}'$ with $|\boldsymbol{S}'| > |\boldsymbol{S}|$. The size of the maximum independent set of graph $\mathbf{G}$ is referred to as the independence number of $\mathbf{G}$ and denoted by $\alpha(\mathbf{G})$ (39).

The edge formulation of the MIS is as follows. Let $\mathbf{x} \in \{0,1\}^{|\mathbf{V}|}$, then

$$\max y = \sum_{i \in V} x_i \tag{4.4a}$$

$$\text{s.t. } x_i + x_j \leq 1, \quad \forall (i,j) \in \mathbf{E} \tag{4.4b}$$

Notice that (4.4b) is the same as $x_i x_j = 0$, provided that x_i is a binary variable. Baring this in mind and following (34) the constraint can be incorporated into the cost-function with a penalty term m. Changing the problem from maximization to minimization, we obtain:

$$\min y = -\sum_{i \in V} x_i + m \sum_{(i,j) \in E} x_i x_j, \tag{4.5}$$

which is equivalent to:

$$\min y = \mathbf{x}^T \boldsymbol{Q}\mathbf{x}, \tag{4.6}$$

where $\boldsymbol{Q} = m\boldsymbol{A} - \boldsymbol{I}$ and $\boldsymbol{I}$ is a $|\mathbf{V}| \times |\mathbf{V}|$ identity matrix.

Pardalos and Xue showed this equivalence for $m = 1$ (40), however, in (11) it is said that penalty parameter can be treated as hyperparameter and optimized further.

Finding the maximal independent set is known to be an NP-hard optimization problem, which means that no exact polynomial time algorithm to solve it exists. Moreover, it was shown to be a strongly NP-hard problem (41) making existence of efficient approximation algorithms unlikely as well.

In this research, I was interested in solving the MIS problem by application of a graph neural network. Since neural networks require continuous loss/cost functions for backpropagation through gradient descent, the binary variable constraint in both Max Cut and MIS problems was discarded. Instead, the solution vector $\mathbf{x}$ was redefined as $\mathbf{y} \in [0,1]^{|\mathbf{V}|}, y_i \in \mathbb{R}$. The binary solution is obtained then as a last step by rounding $\mathbf{x}$ to the nearest integer.

$$\mathbf{x} = \lfloor \mathbf{y} + d \rfloor, \tag{4.7}$$

where d is a decision threshold of choice that in the described experiments was set to 0.5. However, it can be treated as a hyperparameter and be also optimized (11).

4.1.3 Instances

Both MC and MIS problems are defined on graphs. However, more generally, a QUBO model can be viewed as a weighted graph, where an adjacency matrix describes variable interactions, and the weights add information about its strength and direction with respect to the cost function. For the numerical experiments, a collection of graphs of two types was randomly generated with the Networkx library (42): *d-regular graphs* and *probabilistic graphs*. A *d-regular graph* is a graph in which each node has exactly d adjacent edges. To generate a *probabilistic graph*, a random graph model was chosen in which each possible edge appears with probability p, such that $0 < p < 1$.

The graphs of different sizes and densities were studied, in particular for each type of graph a set of graphs $\mathbf{G}(n,d)$ or $\mathbf{G}(n,p)$ was generated, where $n \in \{100, 200, 300, 500, 700, 1000\}$ and $d \in \{3, 4, 5, 6, 7, 9, 10, 15, 17, 18, 19, 20, 21, 22, 23, 24, 25, 30, 35, 40, 45, 50, 60, 70\}$. To keep the comparison between two types of graphs consistent, the parameter p was defined such that the total number of edges in $\mathbf{G}_i(n_j, d_k)$ and expected number of edges of $\mathbf{G}_i(n_j, p_k)$ would be the same (i refers to the i-th graph in the collection, $j \in 1, 2, \ldots, 6$ is the n-th size and $k \in 1, 2, \ldots, 24$ is the k-th density). The expected number of edges in a probabilistic graph is given by:

$$\mathbb{E}_{G(n,p)}(|\mathbf{E}|) = p\frac{n(n-1)}{2}. \tag{4.8}$$

Equating it to the total number of edges in a d-regular graph $\mathbf{G}(n,d)$, which is $\frac{nd}{2}$, we obtain $p = \frac{d}{n-1}$.

4.2 Methods

4.2.1 Baselines

To obtain an unbiased impression of the GNN-solver performance, it was compared to the known classical algorithms that are used to solve MC and MIS. As was mentioned above both MC and MIS problems are NP-hard, thus there are no known polynomial time algorithms. For MIS, even an efficient approximation algorithm does not exist. For these reasons, two heuristics were chosen, proposed in the research and implemented in the Networkx library as of-the-shelf solutions.

Another baseline that was used is gradient descent. Since the GNN-solver utilizes two methods – gradient descent and graph message passing - it is interesting to investigate the contribution of the message passing scheme in the solution process, by comparing it with gradient descent. Specifically gradient descent on the final output of the GNN solver is part of the GNN-solver pipeline as shown in Figure 4.1.

4.2.1.1 Heuristic for the Maximum Cut Problem

The choice was made to utilize a simple one-exchange heuristic as implemented and provided by the Networkx library (43), which tries to locally improve some given cut by one-exchange strategy. The algorithm starts from the cut that partitions a graph vertices into an initially empty subset $\mathbf{S}_1$ and subset $\mathbf{S}_2 = \mathbf{V}$. It then aims to add or swap randomly chosen nodes if this improves the size of the cut. If no improvement can be made, the algorithm terminates. Since the result of the algorithm is dependent on the initialization, it would have been reasonable to perform multiple runs to obtain a better representation of the algorithm's performance. However, the algorithm suffers from long runtimes which for the large graphs become almost prohibitive. Therefore the number of runs was limited to one, which while being not completely representative, in my opinion, still provides fairly reasonable impression of algorithm's performance.

4.2.1.2 Heuristic for the Maximum Independent Set Problem

As was mentioned above, the MIS is a strongly NP-hard problem without known efficient approximation algorithms. The simple minimum degree greedy algorithm (44) was shown to have $\frac{\delta+2}{3}$ performance ratio, where δ is a maximal degree of the graph $\mathbf{G}$, and the performance ratio is defined as follows. Let $A(\mathbf{G})$ be the size of the independent set found by algorithm A and $\alpha(A)$ is a maximal independent set size, then the performance ratio $\rho(A)$ of A is defined by:

$$\rho(A) = \frac{\alpha(A)}{A(\mathbf{G})}. \tag{4.9}$$

The idea of the algorithm is very straightforward: starting from the empty independent set, consequently add one by one nodes from the list of the still available nodes sorted by their degree from lowest to highest. After the node is inserted into the independent set, it and all its neighboring nodes are removed from the list of still-available nodes.

In this research, a modified version of the minimum degree greedy algorithm was used as described in (45) and implemented by the Networkx library (43). In short, the Ramsey algorithm - as it is referred to in the paper - replaces sorting by recursive calls on the neighborhood of a pivot node (the node chosen randomly to be included in the independent set) and its non-neighborhood and choosing the one that attains larger independent set size.

4.2.1.3 Gradient Descent

In order to effectively perform gradient descent on a QUBO cost-function as was mentioned in Section 3.2.1, the cost-function should be continuous and twice differentiable. To meet the continuity condition as was mentioned above, the binary constraint was discarded. The QUBO derivative can be derived as follows. For the sake of an easier notation let me denote $\mathbf{Q}^T = \tilde{Q}$

$$f(\mathbf{x}) = \mathbf{x}^T \boldsymbol{Q}\mathbf{x} = \sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij}x_ix_j \tag{4.10}$$

$$\begin{aligned}\frac{\partial}{\partial x_k} f(x) = \sum_{j=1}^{n} q_{kj}x_j + \sum_{i=1}^{n} q_{ik}x_i = \\ \sum_{i=1}^{n} \tilde{q}_{ik}x_i + \sum_{i=1}^{n} q_{ik}x_i = \\ \sum_{i=1}^{n} (\tilde{q}_{ik} + q_{ik})x_i = \\ \sum_{i=1}^{n} (\tilde{Q} + Q)_{ik}x_i\end{aligned} \tag{4.11}$$

$$\nabla f(x) = (\boldsymbol{Q}^T + \boldsymbol{Q})\mathbf{x} \tag{4.12}$$

Another hurdle to meet is to make sure that $\mathbf{x} \in (0,1)^n$. For this, define $\mathbf{x} = \sigma(\mathbf{z})$, where $\mathbf{z} \in \mathbb{R}^n$ vector and

$$\sigma(\mathbf{z}) = \frac{1}{1+e^{-z}}. \tag{4.13}$$

Then, our transformed version of a cost function to minimize by gradient descent with respect to $\mathbf{z}$ is:

$$\tilde{f}(z) = \sigma(\mathbf{z})^T \boldsymbol{Q} \sigma(\mathbf{z}) \tag{4.14}$$

The vector $\mathbf{z}$ is iteratively updated with the update rule as in (3.11):

$$z^{k+1} = z^k + td^k, \tag{4.15}$$

where t is the step-size or learning rate, $d^k = \nabla_k \tilde{f}(z)$ and

$$\nabla_k \tilde{f}(z) = \frac{\partial f(z)}{\partial \sigma(\mathbf{z}^k)} * \frac{\partial \sigma(\mathbf{z})}{\partial \mathbf{z}^k} = (\boldsymbol{Q}^T + \boldsymbol{Q})\sigma(\mathbf{z}^k) * \sigma(\mathbf{z}^k)(1-\sigma(\mathbf{z}^k)). \tag{4.16}$$

The gradient descent method including all modifications with different optimizers as described in Section 3.2.2.3 was manually implemented from scratch in Python.

4.2.1.4 Analytical solution to the continuous version of $\mathbf{x}^T \boldsymbol{Q} \mathbf{x}$

It should also be noted that the minimum of the QUBO with the relaxation of the binary constraint can be found analytically. To do that a solution to $\nabla f(x) = 0$ or $(\boldsymbol{Q}^T + \boldsymbol{Q})\mathbf{x} = 0$ should be found. It is easy to see that the above expression has only a trivial solution if $\boldsymbol{Q}^T + \boldsymbol{Q}$ is an invertible $|\mathbf{V}| \times |\mathbf{V}|$ matrix. To investigate this, first of all, observe that $\boldsymbol{Q}$ can be transformed into a symmetric matrix (34), by replacing all q_{ij} s.t. $i \neq j$ by $q_{ij} = (q_{ij} + q_{ji})/2$. Then ,

$$\boldsymbol{Q}^T + \boldsymbol{Q} = 2\boldsymbol{Q}. \tag{4.17}$$

The invertibility of the above does not depend on the scaling factor 2, thus it is enough to consider invertibility of $\boldsymbol{Q}$.

Symmetric $\boldsymbol{Q}$ can be converted to the upper triangular form (34) by taking $q_{ij} = 2q_{ij}$ for all $j > i$ and $q_{ij} = 0$ for all $j < i$. Next, according to the Invertible Matrix Theorem, the invertible matrix has a non-zero determinant. The determinant of the upper-triangular matrix is the product of its diagonal entries. Summarizing all the above, it can be concluded that as long as $\boldsymbol{Q}$ does not have any zero diagonal values it is an invertible and continuous expression $\mathbf{x}^T \boldsymbol{Q} \mathbf{x}$ and has its extremum at $\mathbf{x} = \mathbf{0}$.

To find out if this is a minimum, a maximum or a saddle point one might try to analyze eigenvalues of the Hessian matrix of the second derivatives. However, for the present case, it is enough to notice that for both MC and MIS problems, $\mathbf{0}$ or an empty set is never an optimal solution. For both problems, choosing a random node on the graph will construct a better solution with a lower value of the cost function. As Equation (4.3) implies, $\boldsymbol{x} = \mathbf{0}$ will have an associated cost of 0, while including any vertex v_i into the subset $\boldsymbol{S}$ will deliver cost value of $-d_i$, where $d_i > 0$ is the degree of vertex i. Similarly, as following from Equation (6.5) for the MIS problem, independent set $\{v_i\}$ has associated cost $-1 < 0$, the latter being the cost of the trivial solution.

The above illustrates that there is no direct correspondence between binary discreet and continuous on $[0, 1]$ variants of $\mathbf{x}^T \boldsymbol{Q}\mathbf{x}$. The takeaway is that gradient information is not reliable for the QUBO-solving and thus in general it is desirable to avoid convergence to the continuous minimum of the function. However, it is the goal of this research to investigate if the solution process can still be guided by the gradient information and if the GNN-solver is capable of escaping convergence to the trivial solution.

It is known that the plain version of gradient descent is prone to slow convergence, that is why a modified versions are usually used to fasten convergence. In machine learning they are referred as optimizers. Neural networks are almost never trained with plain gradient descent, the optimizer of choice in most cases being Adam, also in (11) Adam was an optimizer of choice. To create a fair experiment setting and investigate how each of these optimizers can help avoid convergence to the continuous minimum, several versions of GD were tested: plain, with momentum, Adam, and Rprop.

4.2.2 Graph Neural Network solver

Typically, the GNN as described in Section 3.2.2 is trained to minimize some loss-function. For the supervised learning tasks the loss-function is usually given by the error measure between the output of the model and given target value. In the unsupervised learning setting, the choice of the loss-function is less trivial, nevertheless the loss function to be minimized can be formulated as some function of the model's output that is being minimized. This setting translates very naturally to the more general optimization approach and thus can be applied for solving QUBOs.

This was the idea that Schuetz, Brubaker, and Katzgraber developed in their paper from 2022 (11). Using the graph interpretation of a QUBO, mentioned in Section 3.1, they proposed to use a GNN on the graph imposed by a QUBO to obtain graph/QUBO representation $\mathbf{h}^k$, where $\mathbf{h}^k$ is a vector of length $|\mathbf{V}|$ after $k - th$ application of the GCN layer. In the code provided with the paper $k = 2$. This representation after application of

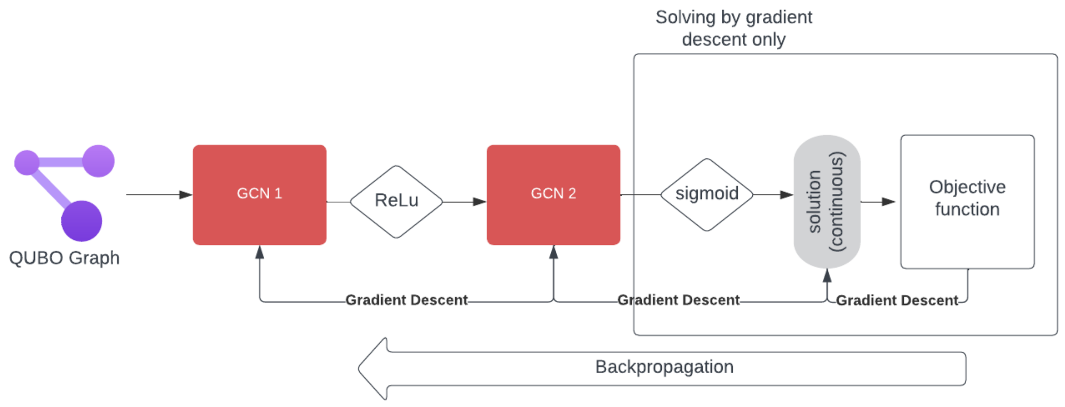


Figure 4.1: Diagram of GNN architecture.

the sigmoid function 4.13 can be interpreted as the probability of binary variable taking value 1:

$$\mathbf{p} = \sigma(\mathbf{h}^k). \tag{4.18}$$

The final integer solution is obtained by taking some discretization function, which in the paper was chosen as simply $int(\mathbf{p})$. The general architecture of the GNN-solver as proposed in the paper is given in Figure 4.1. There are 2 GCN layers as described in 3.2.2.1 implemented by PyTorch (GraphConv layers). Each node in the graph (variable) is provided with a trainable embedding vector of size $\sqrt{|\mathbf{V}|}$. The final output dimension is 1 per node. The learning rate was set to 10^{-4} with the Adam optimizer, the maximal number of epochs to 10^5 with early stopping set to a tolerance of 10^{-4} and patience set to 10^3.

As discussed in Section 5, this architecture suffers from a number of drawbacks. The relatively long solution time is one of them. Further, the solver seemed to perform well only on very sparse graphs. This led to the series of experiments in which the proposed QUBO-solver was modified to broaden the density-range of the solvable QUBOs/graphs. The full list of experiments is given in Table 4.1. Below, I shortly discuss the main adjustments that were made and tested in different combinations. The results of these experiments will be discussed in Section 5.

Change of optimizer from Adam to Rprop In the preliminary series of experiments with Gradient Descent it became clear that Adam is not the best optimizer over all graphs for solving the MIS problem. Adam performs well on the very sparse graphs only. As

Experiments List						
Ref. numb.	Experiment Name	Problem	Solver	Optimizer	Embedding Transfer	N runs
1	Basic MC	MC	GCN-solver	Adam	–	5
2	Basic MIS	MIS	GCN-solver	Adam	–	5
3	GCN with Rprop, MC	MC	GCN-solver	Rprop	–	10
4	GCN with Rprop, MIS	MIS	GCN-solver	Rprop	–	10
5	GCN with Embedding transfer, MIS	MIS	GCN-solver	Rprop	yes	10
6	GD plain, MC	MC	GD	–	–	5
7	GD plain, MIS	MIS	GD	–	–	5
8	GD with momentum, MC	MC	GD	momentum	–	5
9	GD with momentum, MIS	MIS	GD	momentum	–	5
10	GD with Adam, MC	MC	GD	Adam	–	5
11	GD with Adam, MIS	MIS	GD	Adam	–	5
12	GD with Rprop, MC	MC	GD	Rprop	–	5
13	GD with Rprop, MIS	MIS	GD	Rprop	–	5
14	GraphSAGE, MC	MC	GSAGE-solver	Rprop	–	10
15	GraphSAGE, MIS	MIS	GSAGE-solver	Rprop	–	10
16	GraphSAGE, MIS	MIS	GSAGE-solver	Rprop	yes	10

Table 4.1: List of numerical experiments.

the density of a graph was growing, gradient descent with momentum outperformed Adam easily. For the GCN-solver use of Adam optimizer resulted in sometimes considerable runtimes, especially as number of nodes (QUBO variables) grew. This initiated a search for a better optimizer. Rprop was chosen for a considerable solution speed increase, which seemingly was not traded off for the quality of the solution.

Replacing GCN graph convolution module with GraphSAGE The choice of the GCN architecture was indicated by the authors as arbitrary and made based on the frequency of the GCN module usage in research and applications. Another related paper of the same authors (46) uses graph neural networks to solve graph coloring problems on graphs without casting it into QUBO formulation. In the paper, the authors test and compare solvers build on both message-passing architectures: GCN and GraphSAGE. The paper shows that GraphSAGE-solver outperforms GCN-solver at the cost of longer run-times. Since the problem of run-time can be mitigated by the use of Rprop optimizer, in this research GraphSAGE module was also tested against GCN-based solver.

Transferring embeddings trained on the Maximum Cut problem to the Maximal Independent Set problem As was discussed in Section 3.2.2, the output of a graph neural network produces a low dimensional representation for each node in the graph. In this research, this representation is given by a one-dimensional vector $\mathbf{p}$ (see Equation (4.18)). However, since the solution process is performed by back propagation through the GNN layers back to the inputs, it is also possible to define and learn upstream representation - embedding - for a particular graph with the idea that such embeddings learned on a particular problem can be later used to facilitate learning/solution process on a different but related problem defined on the same graph.

Preliminary experiments indicated that the GNN-solver was able to consistently find high-quality solutions for the MC problem. However, this was not the case for the MIS problem. Thus, I wanted to test if embeddings of the nodes trained on the MC problem can be later utilized to facilitate the solution of the MIS problem on the same graph. To this end, each node was assigned a trainable embedding vector of size $int(\sqrt{|\mathbf{V}|})$. This size was chosen as suggested in the original paper (11), but it can be treated as a hyperparameter and optimized in future research.

All GNN-solver modifications were implemented based on an open-source library - PyTorch Geometric.

4.3 Experimental setup and hyperparameters

Since all the tested methods are performing a search in the solution space without guarantee to converge to the global optima, the final solution obtained is dependent on the initialization point. Because of this, each of the experiments from the list given in Table 4.1 was performed multiple times. GNN-solver methods with Adam optimizer as well as gradient descent experiments were run 5 times due to the considerable run-time per graph collection. The GNN-solvers with Rprop optimizer were run 10 times.

Each experiment was performed for a collection of regular as well as probabilistic graphs.

Hyperparameters for the GNN solver, such as learning rate, maximal number of epochs for training, number of layers, choice of hidden activation function, and patience parameters are reproduced from the initial research on the GNN-solver (11) as described in 4.2.2 and did not undergo further tuning.

For gradient descent and its modifications, default values for the hyperparameters proposed in the referenced papers were taken, e.g., $\beta_1 = 0.9$ and $\beta_2 = 0.99$ for Adam optimizer. The learning rate was optimized by trying values $\in \{0.1, 0.05, 0.01, 0.005, 0.001\}$ per gradient descent modification. The full list of final hyperparameters used for gradient descent is given in Table 4.2.

Each gradient descent experiment was given a fixed number of search steps to take 10^4, and then the runtime was measured that was necessary to perform them.

Name	lr	γ	β_1	β_2	ϵ	lr^+	lr^-	Δ_{min}	Δ_{max}
Plain	0.001	–	–	–	–	–	–	–	–
momentum	0.001	0.3	–	–	–	–	–	–	–
Adam	0.01	–	0.9	0.999	e^{-8}	–	–	–	–
Rprop	–	–	–	–	–	1.2	0.5	0.0001	50

Table 4.2: Hyperparameters per gradient descent setup.

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

5

Numerical Experiments Results

This section presents and discusses results obtained with numerical experiments as described in Section 4.

5.1 Gradient descent methods

As was shown in a recent work Bowles et al. (47), gradient descent can achieve high quality solutions in terms of the fraction of known maximal size of the cut for the graph with $|\mathbf{V}| = 2000$. Nummerical experiments in this research have not reached similar levels of the performance due to the fact that gradient descent methods are not the main topic of this research and were used mainly to get better insight into susceptibility of MC and MIS problems to gradient descent methods. These insights are presented below and were used extensively to guide experimentation process with GNN-solvers.

Difference between MC and MIS Figure 5.1 shows a striking difference in performance on MC and MIS problems. It can easily be seen that similar to the results referenced above, plain gradient descent consistently finds cuts of $80 - 90\%$ size of the cut found by classical algorithm, while performance deteriorates very quickly for the MIS problem as the density of the graphs increase. For the graphs with a density of 30 and higher, gradient descent almost always converges to the trivial solution as was described in Section 4.2.1.4.

The possible reason for this difference between the two QUBO problems at hand is the fact that in its original formulation, the MIS is a constrained problem as opposed to MC. To arrive at the QUBO formulation for the MIS, it is necessary to add a penalty term for the objective function as was shown in Equation (4.5). It is possible that the penalty size should be optimized further in future research. However, it should also be noted that all

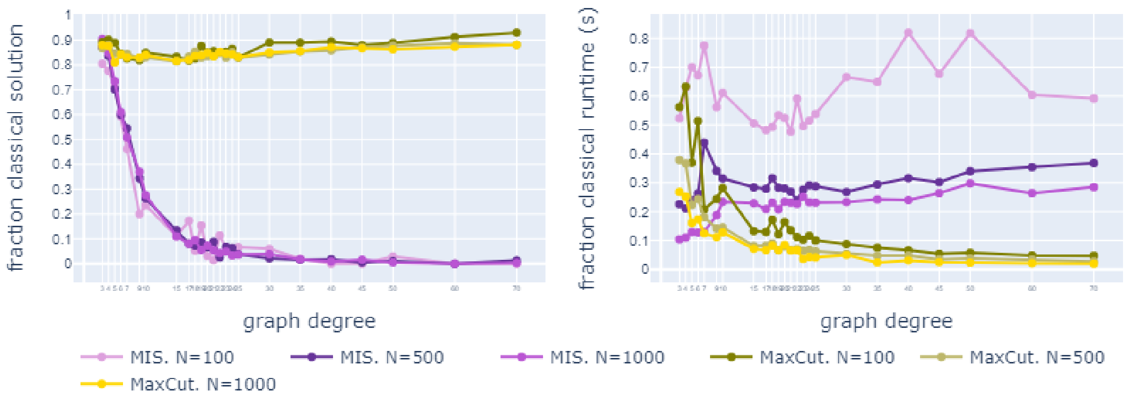


Figure 5.1: Comparison performance of plain gradient descent on Max Cut and MIS problems. Regular graphs. Left: Average fraction of "classical" solution found. Right: Average fraction of classical algorithm runtime spent.

the solutions found by gradient descent for the MIS problem for both types of graphs were always feasible, which indicates that the penalty parameter m is large enough to ensure the correctness of the solution.

Graph connectivity and quality of the solution For the MIS problem, gradient descent can be used for solving only very sparse cases (see Figure 5.1, left). As the density of the graph grows, the quality of the solution found quickly deteriorates.

Difference between regular and probabilistic graphs As Figure 5.2 shows, the quality of the solution for the MC problem found by plain gradient descent is less dependent on the type of the graph than on its size and density. However, this is not true for the MIS problem. It can clearly be seen that when solving MIS problems on probabilistic graphs of comparable density, gradient descent finds slightly better solutions than on regular graphs. Especially on sparse graphs, even plain gradient descent was able to find independent sets larger than the ones found by a classical heuristic. The exact reason for this behaviour is left to future research.

Optimizers Bowles et al. (47) indicated that the plain gradient descent algorithm "has difficulty leaving its initial parameters" and "momentum-assistance is vital in achieving good performance". As can be seen from Figure 5.3, plain gradient descent and gradient descent with momentum assistance outperformed both Adam and Rprop optimizers in terms of the size of an independent set found. Specifically, the dramatic drop in the size

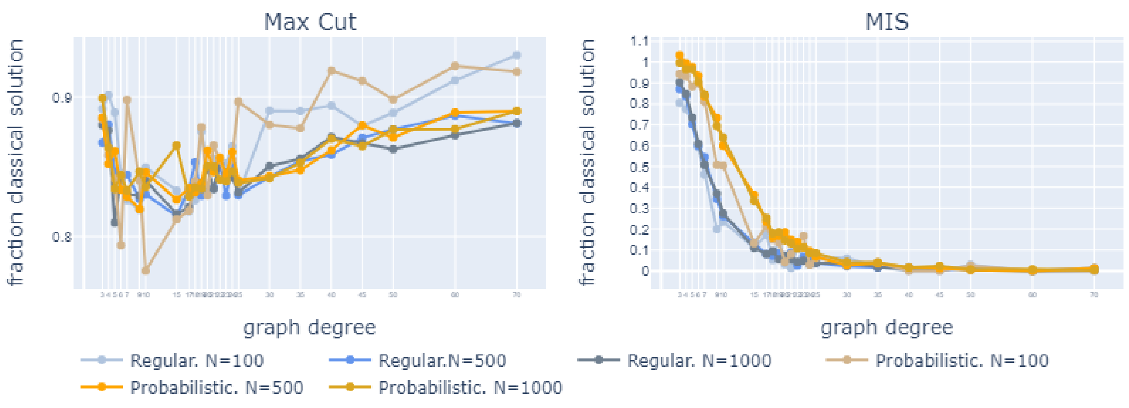


Figure 5.2: Comparison performance of plain gradient descent on regular and probabilistic graphs.

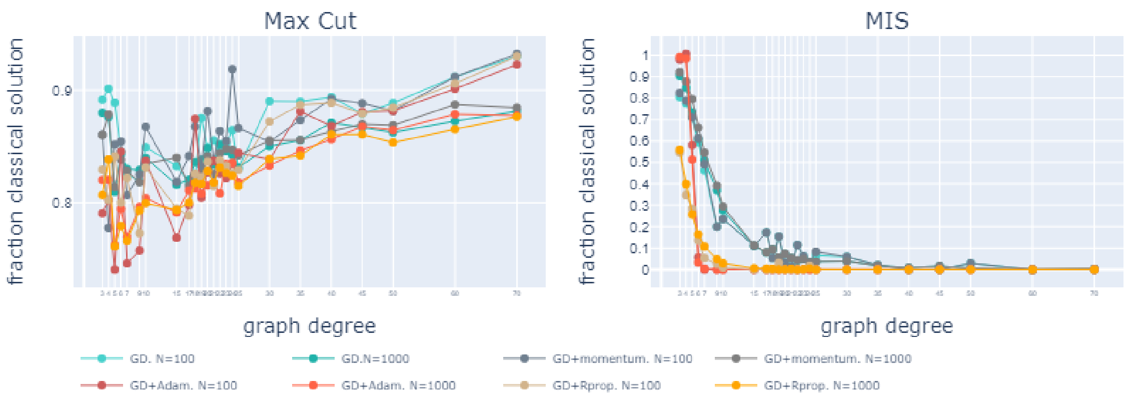


Figure 5.3: Performance per gradient descent method. Regular graphs.

of a set found with the Adam optimizer should be noted. As the right subplot of Figure 5.3 shows, Adam outperforms other gradient descent methods on extremely sparse graphs with $d \in \{3, 4\}$, however, severely underperforms on denser graphs. This is perhaps not a surprising outcome, keeping in mind the analytical solution discussed in Section 4.2.1.4. Adam is often used in practice for the fastest convergence to the local minima. This efficiency in finding local minima, which is in this case a trivial solution is what in fact makes the Adam undesirable optimizer for the problems at hand.

Runtime efficiency Finally, comparing average runtime of plain gradient descent to the runtime of classical heuristics, it can be observed, that for the MC problem gradient descent considerably outperforms classical approach, especially on the dense graphs, see the left subplot of Figure 5.1. However, one should also keep in mind that the one-exchange algorithm for the MC scales very poorly with the size and density of a graph. The greedy approach for the MIS on the other hand displays the opposite behaviour. Since most of its time cost is incurred by sorting nodes that are still not connected to the nodes that are present at the independent set at iteration k, it is clear that for dense graphs, the algorithm will terminate sooner, and the total size of all the sorted sets will be considerably smaller. Still experiments suggest that gradient decent is more runtime efficient than classical heuristics.

To summarize all the above, it was observed that while gradient descent solvers can provide solutions to the MC, though suboptimal, they failed to produce high-quality solutions to the MIS problem for the majority of the graphs. The density and symmetry of the graphs are the obstacles that gradient descent methods struggle against. However, different methods respond differently to the gradual change in graph connectivity. Gradient descent with Adam optimizer showed the most dramatic drop in performance. On the positive side, 100% of the MIS solutions found by gradient solvers were feasible. Also, the runtime was considerably better than the runtime of the classical heuristics.

5.2 Graph Neural Network solvers

In the previous section it was discussed that the performance of gradient descent methods is different per problem, that is why performance of the GNN-solvers is discussed below per problem.

5.2.1 Max Cut

As can be seen from Figure 5.4 the basic version of the GNN-solver as implemented in (11) produces high-quality solutions for the Max Cut problem for all graph sizes and densities both regular and probabilistic.

Further it can be noted, that for many graphs, the basic GNN solver, on average found larger sizes of the cut than the classical heuristics. For the large graphs, it also demonstrated significant speedup compared to the classical heuristics (see Figure 5.5 (a), (b)).

Replacing the Adam optimizer with Rprop reduced the runtime, but the solution quality deteriorated slightly, as shown in Figure 5.5. Notice, however, that due to the considerably reduced solution time, it is possible to make more solution runs with Rprop than with Adam within the same computational time budget. This is a reasonable thing to do, since these methods are dependent on initialization and multiple runs allow them to search through larger areas in the solution space. Specifically, it was observed that the best cut

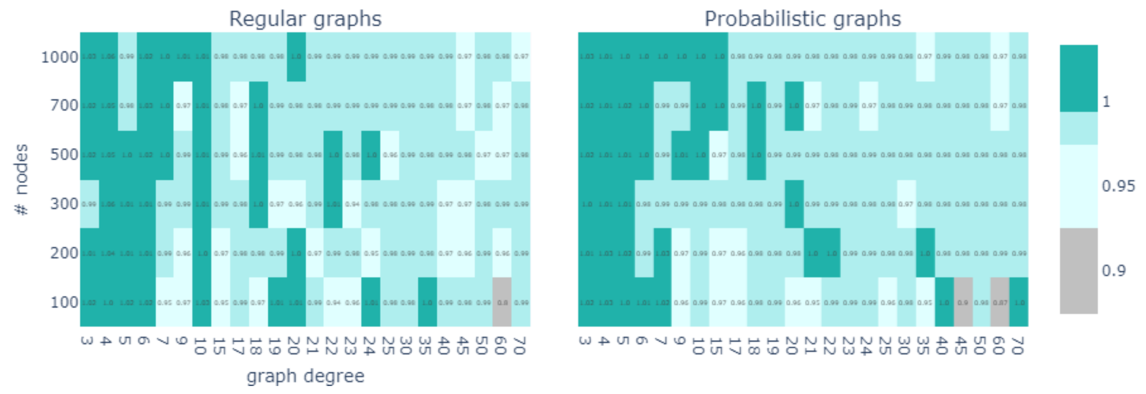


Figure 5.4: Average fraction of "classical" solution found by the basic GNN solver.

found by the GNN solver with Rprop out of 10 runs was larger than the average cut found by the GNN solver with Adam in 5 runs.

The overall conclusion from this series of experiments is that Max Cut problem can be solved fast and with reasonably high quality of the solutions by the basic GNN solver as described in (11). Adding the Rprop optimizer to the architecture might be beneficial in some cases to be able to perform a broader search over solution space within the same time budget.

5.2.2 MIS

It became obvious from the first round of experiments with the GNN-solver implemented as described in (11), that the ability of that solver to find high-quality solutions is limited to only very sparse graphs. As can be seen from Figure 5.6, for graphs with $d > 7$, the solver always returned an empty set as a solution. The very similar performance of gradient descent with the Adam optimizer was already discussed in 5.2 (see Figure 5.3, right subplot).

Thus the first modification added to that architecture was an optimizer change from Adam to Rprop which resulted in considerable reduction of training time combined with the increased ability to solve denser graphs as shown in Figure 5.7. The GCN-solver with Rprop optimizer was able to produce high-quality solutions for graphs with $d \leq 10$ within reduced solution time. Further, all obtained solutions were feasible for both types of graphs without any violations of the independence constraint. That is why the rest of the experiments with GCN-solver modifications were performed with Rprop optimizer.

In section 5.2, it was also observed that regular graphs were less susceptible to the gradient descent methods compared to probabilistic graphs, probably due to the symmetry

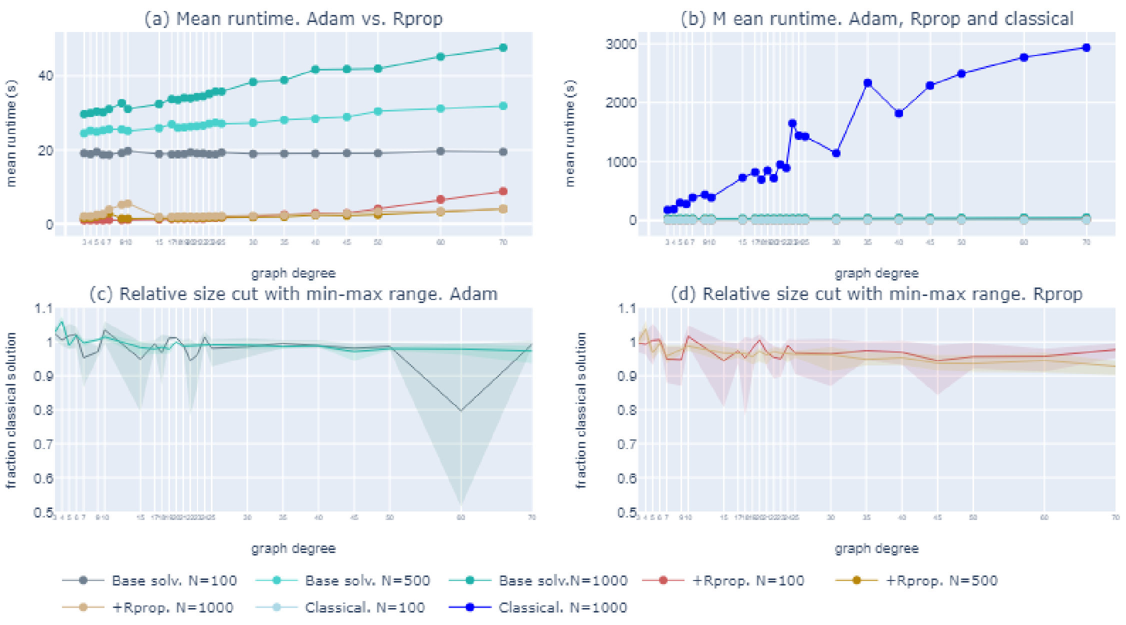
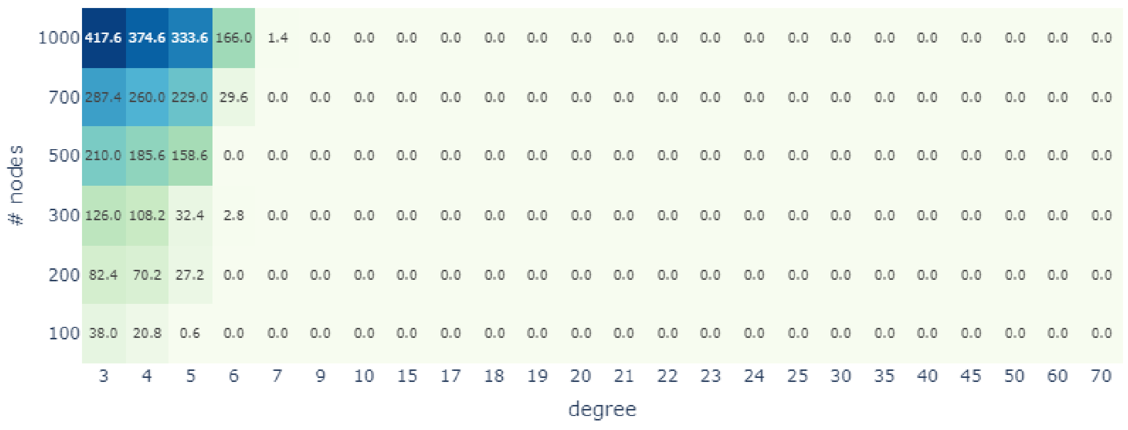


Figure 5.5: Max Cut. Regular graphs. Comparison of the baseline GCN-solver with the Adam vs. Rprop optimizer.



# nodes	3	4	5	6	7	9	10	15	17	18	19	20	21	22	23	24	25	30	35	40	45	50	60	70
1000	417.6	374.6	333.6	166.0	1.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
700	287.4	260.0	229.0	29.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
500	210.0	185.6	158.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
300	126.0	108.2	32.4	2.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
200	82.4	70.2	27.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
100	38.0	20.8	0.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 5.6: Average size of the independent set found by the basic GNN solver on regular graphs.

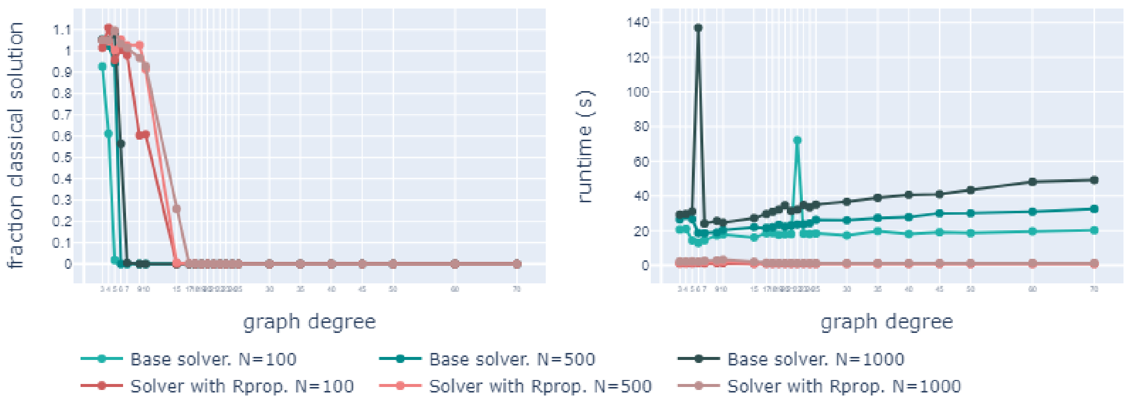


Figure 5.7: Comparison of the Adam vs. Rprop optimizer. GCN-solver. Regular graphs. Left: Average fraction of "classical" solution found. Right: Average runtime.

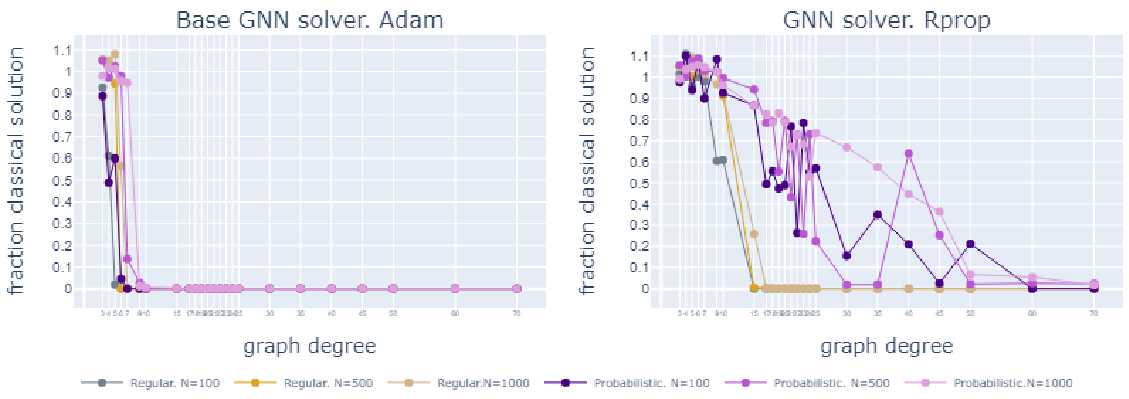


Figure 5.8: Difference in performance per graph type for GNN solver with Adam and with Rprop.

of the former. As figure 5.8 shows, for the default version of the GNN-solver with Adam, there was a very slight difference in performance per graph type. However, the GNN-solver with Rprop showed considerable performance improvement for probabilistic graphs compared to the regular graphs of the same density.

Possibly, this difference in performance per graph type is caused by the difficulty of obtaining potentially useful global information about the graph from regular and thus symmetric graphs. This suggests that if information about the global graph landscape can be learned, this information might facilitate a solution process and guide it on dense graphs where gradient descent information fails.

However, experiments indicated that learning them at the same time as solving the MIS problem was not always possible for a GCN-based solver. Thus, the idea emerged to train

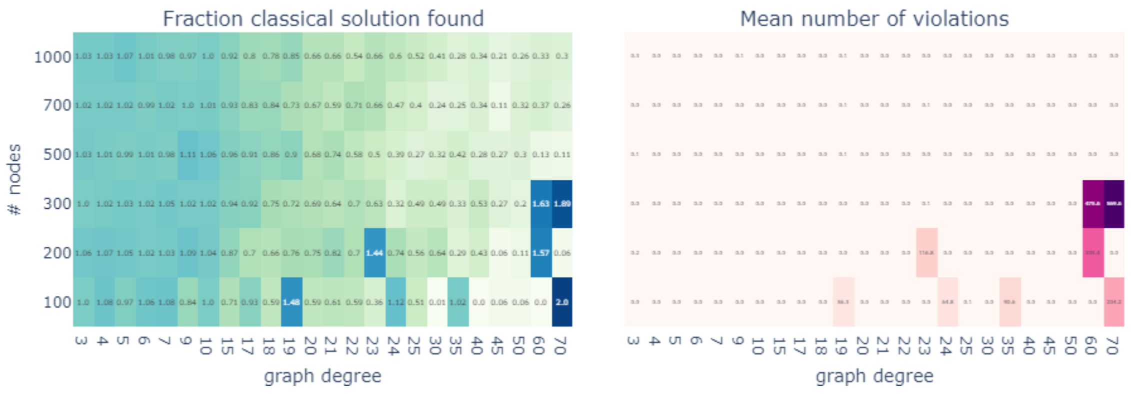


Figure 5.9: GNN-solver with embedding transfer. Regular graphs

graph embeddings on the MC problem, since, as was shown above, it is solvable by the GCN-solver for any graph, irrespective of their density. The hypothesis is that although the MC and MIS problems are not directly related, it is to be expected that the vertices subsets $\mathbf{S}_1$ and $\mathbf{S}_2$ returned as a solution to the MC would have less inner connections, and will be more independent than just randomly sampled. Thus, node embeddings learned on the MC problem might also guide a solution process for the MIS problem on the same graph.

The results of this experiment tentatively suggest that this was a valid hypothesis. For the regular graphs the range of graphs in terms of their degree for which a quality solution can be obtained expanded considerably with a fraction of total runs in which an empty set was returned as a solution going to zero for most of the graphs. However, as Figure 5.9 shows, the feasibility of the returned solution is not guaranteed, especially on the dense graphs a lot of violations of the independence constraint were detected.

Figure 5.10 shows an increase in the size of an independent set returned for some denser regular graphs. However, this performance improvement comes at the price of increased run-times, since the solution time required for solving one particular instance of a MIS problem includes also a solution time for the MC problem on the same graph instance. For the probabilistic graphs the performance increase is less pronounced which was to be expected, since the GCN-solver was able to gather some global graph information even without the embeddings transfer.

When performing an embedding transfer, as described in Section 4.2.2, one has an option to freeze them, which will exclude them from the training process to solve the MIS. Frozen embeddings are treated like constant input and are not adjusted in the training loop. Another option is to unfreeze them and continue training for a problem at hand.

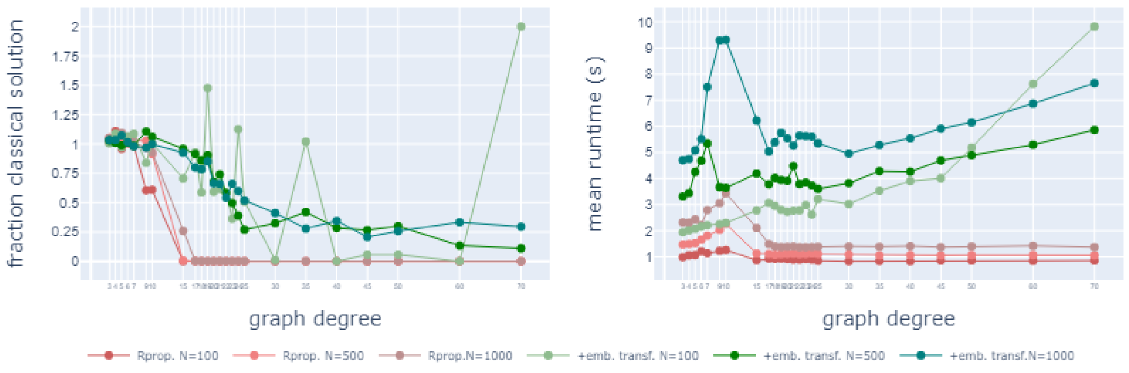


Figure 5.10: GNN-solver with and without embedding transfer. Regular graphs. Left: Average fraction of "classical" solution found. Right: Average runtime.

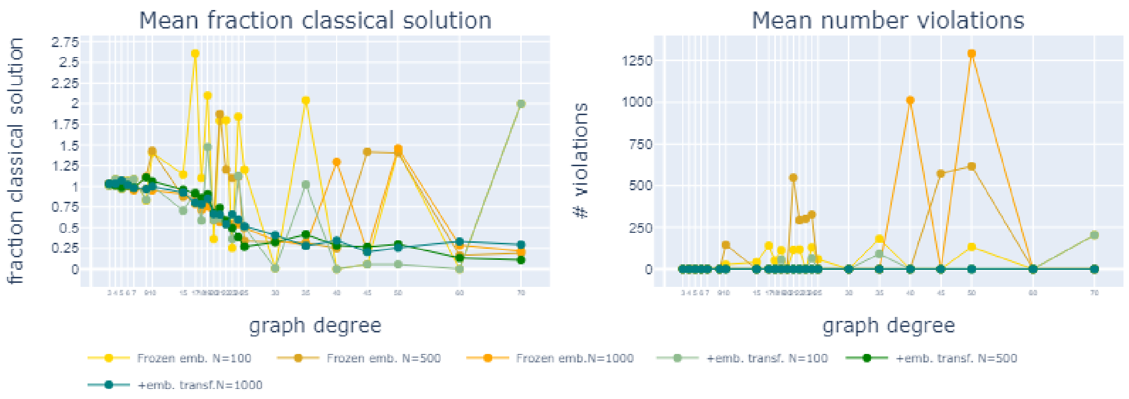


Figure 5.11: Comparison of the GNN-solver with frozen and not frozen embeddings transfer. Regular graphs.

Both approaches were tested. From the first glance at Figure 5.11, left subplot, frozen embeddings might seem to deliver larger independent sets. However, the right subplot reveals that the higher cardinality of sets returned as solutions is caused by a considerable number of violations of the independence constraint. Thus, solving the MIS with frozen embeddings initially trained on the MC instance consistently has a higher chance of obtaining an incorrect solution, which is to be expected, since both MC and MIS problems are not immediately related. Direct application of the MC embeddings thus might be misleading in some cases. When those embeddings are not frozen, they are able to guide the search process while being simultaneously adjusted to a different cost function. Also, Figures 5.9 and 5.10 suggest that the embedding transfer works particularly poor on very dense graphs, producing solutions with considerable number of violations. It is also likely that embeddings trained on the MC problem for the particularly dense graphs do not contain any information that can help the solver to infer which vertices are in fact independent. This is due to the fact that as the density of a graph grows, subsets $\boldsymbol{S}$ and $\boldsymbol{S}'$ returned as a solution to the MC problem, contain a progressively higher number of inner edges and these subsets are less independent than the ones found on the sparse graphs.

To summarize the above, although the embedding transfer might provide some useful information to the GNN-solver in some cases, this approach combined with the GCN architecture still cannot completely overcome the density hurdle.

The final modification added to the GNN-solver was replacing the GCN module with the GraphSAGE. The results of this experiment are shown in Figure 5.12. It can easily be seen that out of all modifications tested, the GraphSAGE-solver produced the best results on large graphs. The solutions found by the GraphSAGE-solver are closest in terms of cost function value to the solutions found by classical heuristics. The average number of violations is also low, which makes it possible to restore correctness by removing one of the connected nodes from the set returned as a solution. Interestingly, the GraphSAGE architecture did not benefit further from the embedding transfer, which might suggest that it is efficient enough in learning useful node representations during the solution time. The fact that this separate step of learning embeddings on the MC problem is not required for GraphSAGE resulted in shorter training times as well.

It should also be noted that reasonable performance on the denser graphs comes at the cost of occasional suboptimal performance on very sparse graphs on which other methods excelled. This might suggest that there is no architecture that can handle sparse and dense graphs with equal optimality. There seem to be a trade-off – the ones that produce high quality results on sparse instances fail quickly on dense ones. As was already mentioned, the search process in GNN-solvers is guided by two processes - gradient descent-based backpropagation of the cost function and aggregation of graph information through message passing. The results of all the experiments described above allow for a tentative suggestion that these two sources of direction might conflict and misguide the solution process. The GraphSAGE-solver exhibits the best balance in that regard. Finally, it should be kept in mind that despite the fact that the GraphSAGE solver found high-quality results on larger graphs, the solutions found on denser smaller graphs: $n = 100, d > 20$ were still very far from optimal in terms of cost function. At the same time the quality of the solutions on the larger graphs still slowly deteriorated with the growing density, which indicates that GraphSAGE-solver also can not completely overcome the problem of finding high quality solutions on dense graphs.

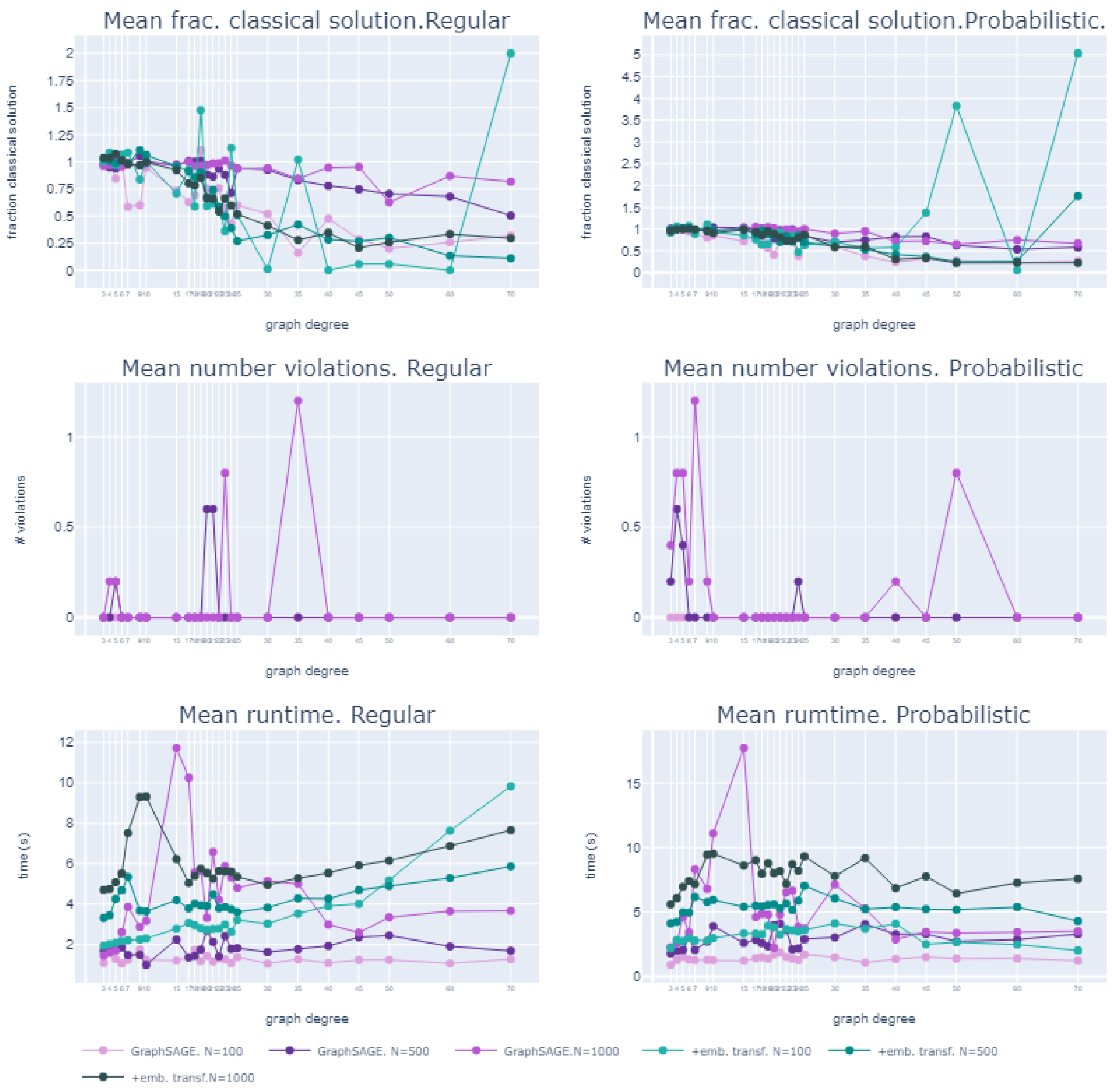


Figure 5.12: GraphSAGE-solver vs. GCN-solver with embedding transfer

6

Portfolio optimization using GNN-solver

6.1 Portfolio Optimization

6.1.1 Problem description

As defined in (48): "portfolio selection in the broadest sense refers to investment of savings in an expectation of higher earnings in future by carefully choosing the right mix of assets". The typical setting for this type of problem involves some set of financial assets (such as shares, bonds, and other financial instruments), each of which can be described by the expected return and the variance of this return, which is usually referred to as a risk of an asset, as it represents the uncertainty of this return in the future (49). The goal is to select from this set a subset of assets subject to some utility function. The main approach to portfolio selection, known as mean variance portfolio theory, was first formulated in 1952 by Harry Markowitz. He stated in (50) that an "investor considers expected return as desirable thing and variance of return as undesirable thing," and classical Markowitz analysis aims at maximizing returns for a given level of risk or minimizing portfolio risk with a threshold return in mind. This approach exposes the multi-objective nature of the problem and thus the inherent trade-off between these two goals. In the following decades, the classical approach was extended to dynamic portfolio selection that deals with multi-timestep optimization; real financial market constraints such as transaction costs or trading rules constraints were included; special methods were applied to reduce the influence of return estimation errors. The comprehensive description of those extensions to the classical theory as well as proposed methods is given in (51).

Here, the GNN-solver is being benchmarked on the portfolio selection task against the results obtained by Frank Phillipson and Harshil Bhatia with the D-Wave Hybrid Quantum Annealer as presented in (52). For this purpose, the portfolio optimization problem is first formulated as in the paper referenced above (Model 1). However, since the obtained results were distinctly suboptimal (see Subsection 6.5.1), the portfolio selection problem was reformulated to the more classical Markowitz form (Model 2) that also demonstrates strong similarity to the MIS problem discussed in Section 4.1.2.

6.1.2 QUBO formulation

Given is a set of assets $\{A_1, A_2, \dots, A_N\}$. For each asset A_i, a return on investment is a random variable:

$$K_i = \frac{S_i(1) - S_i(0)}{S_i(0)}, \tag{6.1}$$

where $S_i(0)$ is the current price of an asset and $S_i(1)$ is a random variable describing an unknown future price of the asset (49). The asset A_i thus can be described by an expected return on investment $\mu_i = \mathbb{E}(K_i)$ and risk expressed in terms of σ_i^2 - the variance of K_i and Pearson correlation coefficient ρ_{ij} with A_j. Variances and correlation coefficients together form a risk matrix Σ.

A "yes"/"no" decision, represented by a binary variable, is being made about the selection of each asset for the portfolio. In the classic Markowitz model the goal is to minimize risk of the selected assets such that the return of the portfolio is at least R (Model 1). The cost function proposed in (52) (Model 1) also introduces "budget" requirement, namely the size of the portfolio (number of selected assets) be equal to n.

These lead to the following formulations.

Model 1. Let $\mathbf{x} \in \{0,1\}^{\mathbf{N}}$, then

$$\begin{aligned} \min \quad & \mathbf{x}^T \Sigma \mathbf{x}, && (6.2a)\\ \text{s.t.} \quad & \sum_{i=1}^{N} x_i = n, && (6.2b)\\ & \mu^T x \geq R. && (6.2c) \end{aligned}$$

The QUBO formulation of this quadratic, constrained, binary model is derived in (52) and is given as follows:

$$\min(\lambda_0 \mathbf{x}^T \Sigma \mathbf{x} + \lambda_1 (\sum_{i=1}^{N} x_i - n)^2 + \lambda_2 (\mu^T x - R - \sum_{k=1}^{K} 2^k y_k)^2), \tag{6.3}$$

where $y_k (k = 1, \ldots, K)$ is a slack variable introduced to incorporate the inequality constraint 6.2c into the cost function and $K = \lfloor log_2(\sum_{i=1}^{N} \mu_i) \rfloor$.

λ_0, λ_1, λ_2 are hyperparameters to be specified.

The first term in Equation 6.3 represents the total risk of the selected portfolio, the second represents the deviation from the required size n, and the last one incorporates the inequality constraint 6.2c referring to the minimal required return of the portfolio R.

Model 2. For the simplified version of the portfolio selection QUBO, the budget constraint was eliminated. Since R represents a shift by a constant along the cost function axis to some extent, it can be absorbed by the weight on the return term λ (introduced below), which allows to simplify QUBO formulation further. That leads to the following formulation.

Let $\mathbf{x} \in \{0, 1\}^{\mathbf{N}}$, then

$$\min(\mathbf{x}^T \Sigma \mathbf{x} - \lambda \mu^T x), \tag{6.4}$$

where $\lambda \geq 0$ is a hyperparameter to be determined upfront. This formulation is very similar to the MIS formulation presented in Section 4.1.2. For the MIS problem $\boldsymbol{Q} = m\boldsymbol{A} - \boldsymbol{I}$ and $\boldsymbol{I}$ is a $|\mathbf{V}| \times |\mathbf{V}|$ identity matrix. For the portfolio selection case, the weighted adjacency matrix $m\boldsymbol{A}$ is replaced by the covariance matrix Σ and identity matrix $\boldsymbol{I}$ is weighted by the expected returns on investment of each asset $\boldsymbol{I}\mu$. Thus, 6.4 is equivalent to:

$$\min(\mathbf{x}^T \boldsymbol{Q} \mathbf{x}), \tag{6.5}$$

where $\boldsymbol{Q} = \Sigma - \lambda \boldsymbol{I} \mu$ and $\boldsymbol{I}$ is a $N \times N$ identity matrix.

6.2 Method

The portfolio selection was performed by the GNN-solver as described in Section 4.2.2 with the GraphSAGE convolution module and the Rprop optimizer.

The results obtained by the GNN-solver were compared to the results presented in (52) that were obtained with the D-Wave hybrid solver that leverages the capabilities of the D-Wave Quantum Annealer to enhance the performance of the state-of-the-art classical optimization algorithms such as Tabu Search.

D-Wave Quantum Annealer Quantum computing is a relatively recent field that has been developing quickly in recent decades. It leverages the unique properties of quantum bits (qubits). The qubits similarly to the classical bits can be in 0/1 state. However, their potential computing advantage is that they can also be in a superposition of the 0 state and the 1 state at the same time (53). There are two distinctive main paradigms for quantum computations: gate model and quantum annealing. Gate model computers perform computations through the series of quantum gates that are similar to the binary gates on the classical computers. Quantum annealers on the other hand use a different approach where the optimization problem formulated as an Ising model is physically embedded on the quantum device where each decision variable is represented by one or multiple qubits.

The Ising model has the following form. Given the graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, with vertices weights h_i and edge weights J_{ij}, a spin $x_i \in \{-1, 1\}$ has to be assigned to each vertex with the objective:

$$\min(\sum_{i \in V} h_i x_i + \sum_{(i,j) \in E} J_i x_i x_j), \tag{6.6}$$

Obviously, any QUBO problem can be directly mapped to the Ising model by using $z_i = 2x_i - 1$.

The quantum annealer solves the offered problem instance by finding lowest energy state for the Issing model. Annealing is a process of gradual transformation of the initial Hamiltonian to the Hamiltonian that encodes the input optimization problem, which can be expressed as follows:

$$H_{issing} = -\frac{A(s)}{2}(\sum_i x_i) + \frac{B(s)}{2}(\sum_i h_i x_i + \sum_{i>j} J_i x_i x_j), \tag{6.7}$$

where the first term is initial Hamiltonian in low energy state and the second term is the final Hamiltonian describing the problem to solve. The A and B are dependent on time and represent the annealing schedule. During the annealing phase, the problem Hamiltonian is slowly introduced, and the influence of the initial Hamiltonian is reduced. "If the system evolves very slowly (i.e., adiabatically), it will eventually settle in a final ground state that, with a certain probability, will correspond to the optimal value of the function" (54). Ideally, the system stays in the minimum energy state throughout the quantum annealing process and arrives to the minimum energy state of the problem Hamiltonian and therefore has an answer to the problem at hand. By the end of the anneal, each qubit is a classical object (53).

Currently, quantum computing is still in the development stage and pure quantum computations are feasible only for relatively small problem instances. For gate-based quantum computers, the most recent advancement was presented by IBM. Their Osprey processor has 433 qubits. The D-Wave quantum annealer operates with 5000 qubits (52). For the real-life sized problem instances, hybrid solvers are offered. The exact implementation details and used algorithms are not disclosed by D-Wave, however they claim that their solvers "implement state-of-the-art classical algorithms together with intelligent allocation of the quantum computer to parts of the problem where it benefits most" (55).

6.3 Experimental Setup

The work on the portfolio selection with the GNN-solver was structured as follows.

Experimental Setup 1: the experimental setting of Phillipson and Bhatia was exactly reproduced, with sub-experiments and respective λ_0, λ_1, λ_2 as defined in Table 1 of (52), where each experiment represents a combination of N-first assets taken from the Nikkei225 index to select a portfolio from, n - size of the portfolio and R - required minimal return of the portfolio. It must be noted that in the original paper, all the expected returns were transformed to their absolute value and multiplied by 100. The risk matrix was multiplied by 1000. This transformation was preserved in this setup for the sake of comparison of results.

Experimental Setup 2: As an attempt to improve performance of the GNN-solver, λ_0, λ_1, λ_2 were reoptimized to be used with Model 1.

Experimental Setup 3: Model 2 was used for portfolio selection.

Experimental Setup 4: A Pareto frontier of optimal portfolios was derived based on the Model 2 formulation.

Since the GNN-solver result depends on initialization, all experiments were repeated 50/100 times depending on the experimental setup.

6.4 Data

Following Phillipson and Bhatia, the portfolio selection was performed on Nikkei225 index which is a stock market index for the Tokyo Stock Exchange. It is a price-weighted equity index that includes 225 stocks of the most important Japanese companies. For the sake

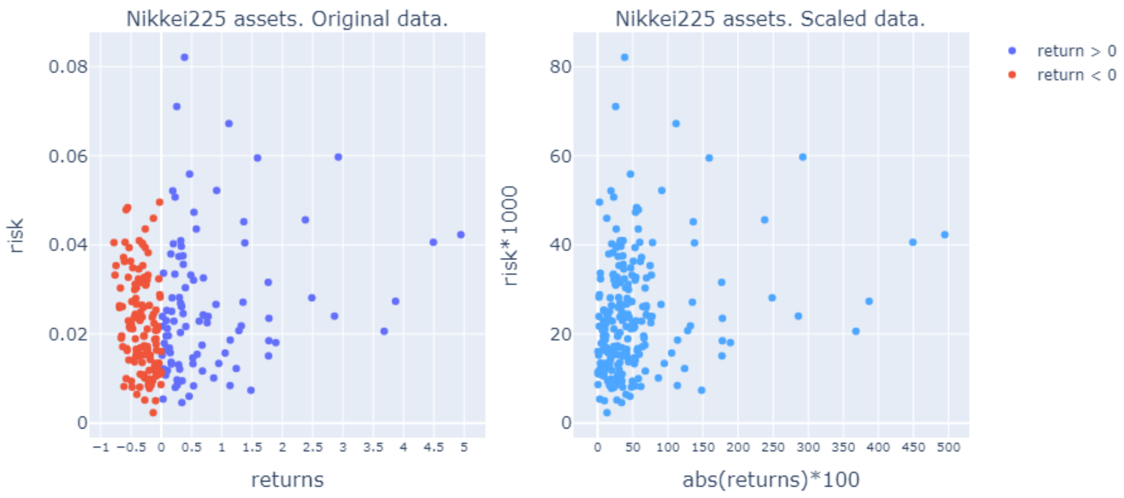


Figure 6.1: Nikkei225 assets. Left: Original data Right: Transformed data.

of consistent comparison the selection was performed on the same data as in (52), which consists of a 5-year return on investment calculated over the period from 2015 to 2020, which estimates $\mu_i = \mathbb{E}(K_i)$, and variances and covariance of the quarterly returns over the same period that are used to estimate covariance matrix Σ.

As was mentioned in Section 6.3, in an Experimental Setup 1, transformation was performed on the initial returns and risks. The comparison of initial and transformed data is shown in Figure 6.1.

It can easily be seen that there are quite some outliers among the assets in terms of unusually high returns or significant variance. Those outliers cause a considerable skewness in the distribution of asset returns. Markowitz mean-variance portfolio selection theory assumes normal distribution of returns and this assumption is not met as demonstrated by Figure 6.2. This should be kept in mind when deciding on the practical relevance of the obtained results. However for the sake of investigation and measuring of the GNN-solver performance this data issues do not pose an obstacle.

Next, it is interesting to see if the imposed minimal returns R are feasible within portfolio sizes defined in the paper and can they be achieved by random selection (52). Figure 6.3 shows that required returns are difficult to reach by random selection of the assets for the small sized portfolios, however, for the larger portfolios the required minimal returns do not pose a challenge and can be achieved by random selection. Secondly, it can also be seen that obtaining the required return is easier on the specific subsets than on the others. For example, the left subplot of Figure 6.3 suggests that limiting the set of the assets

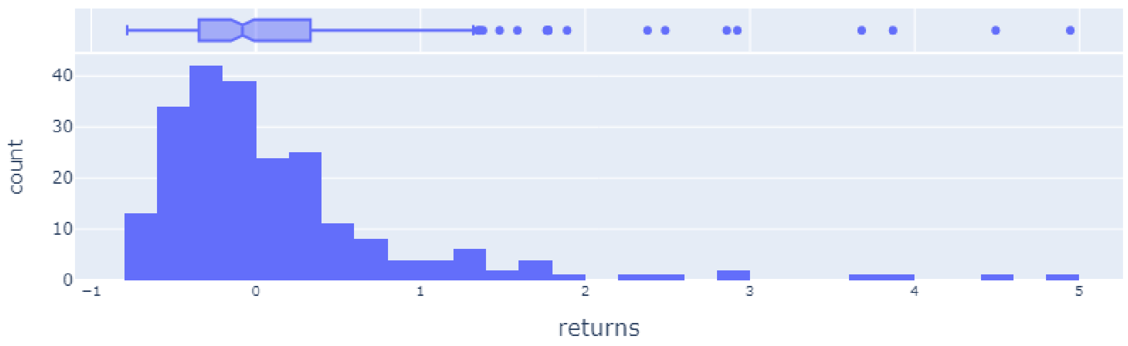


Figure 6.2: Histogram of Nikkei225 assets original returns.

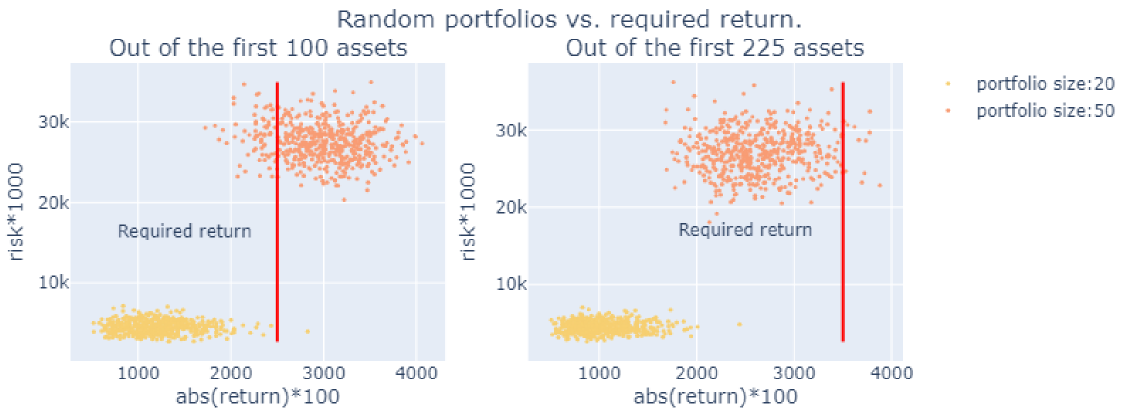


Figure 6.3: Returns and risks of 500 randomly selected portfolios of sizes 20 and 50 compared to minimal required return R.

to be selected from (N) to only the first 100 items most likely excludes some obviously undesirable assets before the actual selection process, which reflects in the fact that the majority of the randomly selected portfolios easily meet minimal return requirement R. Therefore, some preselection of the assets might be desirable.

6.5 Results

6.5.1 Experimental Setup 1

When minimizing the cost function defined in Phillipson and Bhatia (Equation 6.3) GNN-solver delivered low quality solutions.

First of all, the "budget constraint" was almost never met by the solver. As Figure 6.4 shows, the size of the portfolios selected by the GNN-solver seems to be distributed around some mean size that changes its location per experiment parameters combination

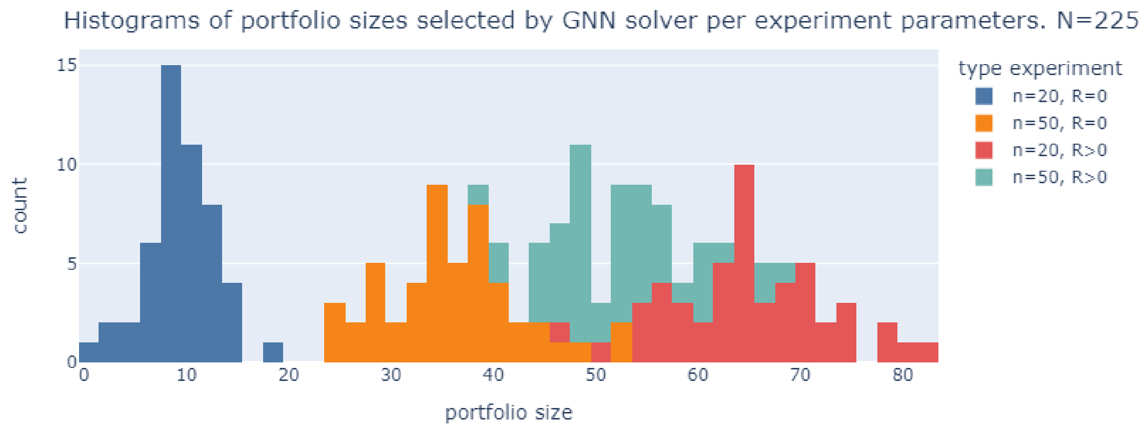


Figure 6.4: Histograms of portfolio sizes selected by GNN solver per experiment parameters. $N = 225$

as proposed in Phillipson and Bhatia, Table 1 (52). However, it does not seem to have an immediate relation with the required size of the portfolio n.

Second, the quality of the derived portfolios was mostly the same as per random selection. Figures 6.5 and 6.6 show the results of the portfolio selection performed at random and by the GNN-solver with parameters N and n being equal. In those figures, portfolios selected by the GNN-solver are plotted against 100 randomly selected portfolios of the same size. It can clearly be seen that when no minimal return is required, solutions returned by the solver are of the same quality as derived by random selection. If the required return $R > 0$ is specified, portfolios that are significantly larger in size (despite the required size being 20) are selected, with better than random return but usually with a higher risk as well.

Since in this experimental setting, experiments performed in (52) were reproduced exactly, including the same lambda weights for the cost function as derived in the paper for D-Wave Quantum Annealer, the next step was to attempt to reoptimize lambdas for the GNN-solver.

6.5.2 Experimental Setup 2

It is generally recommended to scale data when working with neural networks (56). That is why it was decided to abandon data transformations introduced in (52) and work with the original data scaled by min-max transformation:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}. \tag{6.8}$$

λ_0, λ_1, λ_2 are hyperparameters used in the cost function as defined by Equation (6.3). Their optimization was performed through the grid search. The potential set of values for

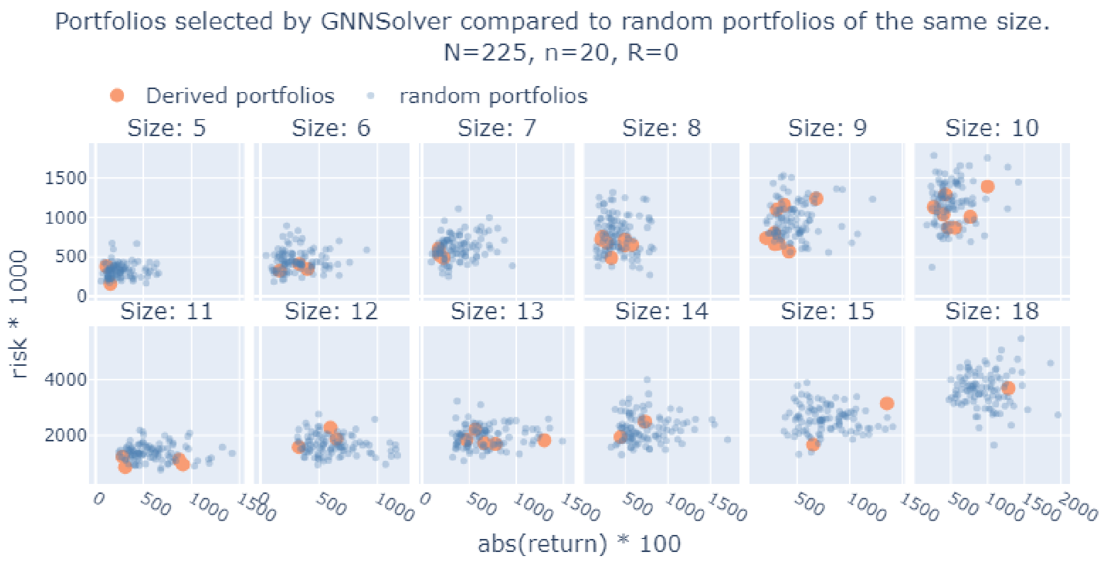


Figure 6.5: Portfolios selected by GNNSolver compared to random portfolios of the same size. $N = 225$, $n = 20$, $R = 0$

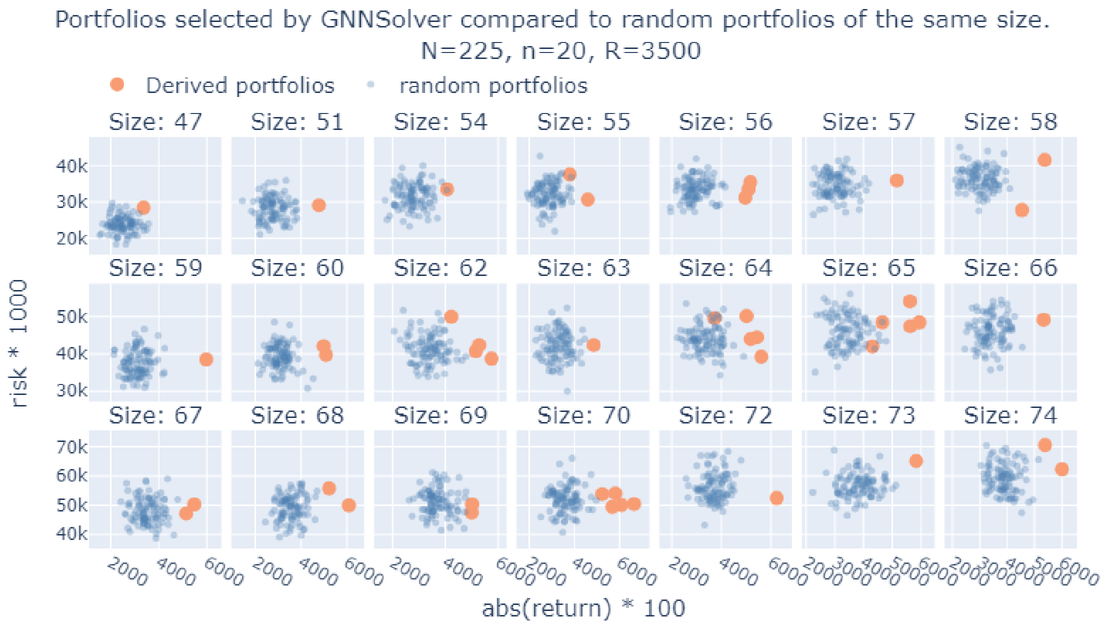


Figure 6.6: Portfolios selected by GNNSolver compared to random portfolios of the same size. $N = 225$, $n = 20$, $R = 3500$

λ_0 and λ_2 was defined as $\{0.1, 1, 5, 10\}$. The "budget constraint" term was unresponsive to the change of values within this range, which is why values for λ_1 have higher magnitudes: $\{1,\ 100,\ 1000\}$. In total, this delivered a grid of 48 possible combinations. As the second step two special fine-tuned combinations were added: $\lambda_0 = 2$, $\lambda_1 = 1$, $\lambda_2 = 10$ and $\lambda_0 = 3$, $\lambda_1 = 1$, $\lambda_2 = 10$.

For each combination of hyperparameters, 50 solution runs were made by a GNN-solver. The goal of this reoptimization was to determine whether it is possible to recalibrate cost function (Equation (6.3)) such that the GNN-solver will improve its performance in terms of the quality of the solutions found. Therefore, reoptimization was performed for only one set of experiment parameters, namely $N = 225$, $n = 50$, $R = 35$, as a try-out. For the sake of the easier analysis of obtained results, all possible combinations of the parameters can be clustered into 6 groups:

1. parameter combinations where λ_0 dominates (risk minimization is preferred),
2. parameter combinations where λ_1 dominates ("budget" constraint is preferred),
3. parameter combinations where λ_2 dominates (maximization of return is preferred),
4. parameter combinations where λ_0 and λ_2 dominate ("budget" constraint is suppressed),
5. parameter combinations where $\lambda_0 = \lambda_1 = \lambda_2$,
6. 2 fine-tuned combinations added at the second step.

Note however that this division describes only dominance of the objective directly imposed by λ_0, λ_1, λ_2. The dominance of the objective that can be caused by the difference in scaling of the data used per objective is left out of this division, since the purpose was to investigate the influence of λs.

Figure 6.7, left part, shows returns, risks, and sizes of the portfolios selected by the GNN-solver averaged over 50 runs per combination of hyperparameters. As expected, increasing weight on the risk minimization term (λ_0) leads to smaller portfolios with lower risk but also lower returns. Emphasizing return term leads to larger portfolios with higher returns but also higher risk. The worst portfolios (almost all of them are of the random quality) are produced by combinations of hyperparameters with dominating "budget constraint" term. On the other hand suppressing "budget constraint" term leads to better solutions (see hyperparameters groups where λ_0 and λ_2 dominate and fine-tuned combinations). This

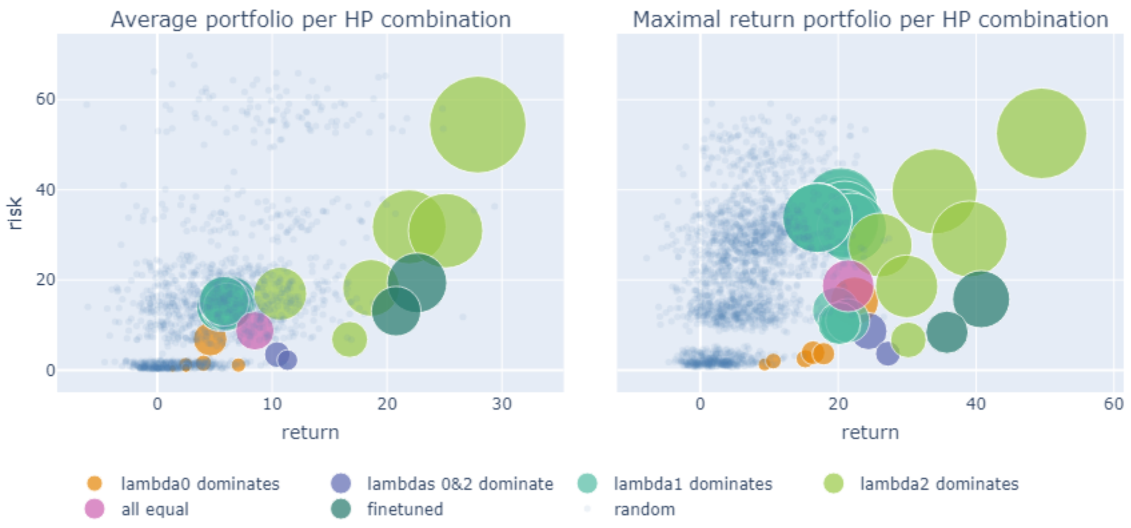


Figure 6.7: Average and best portfolios per hyperparameter combination compared to 100 randomly selected portfolios of comparable sizes. The dot sizes of the solved portfolios correspond to their respective sizes. Random portfolios are plotted without sizes.

effect is even more pronounced when inspecting portfolios with the highest revenues per hyperparameter group displayed in Figure 6.7 Right part, where fine-tuned group seems to form a Pareto frontier of sorts.

Figure 6.8 summarizes distinctive features by a hyperparameter group. The plot confirms again that putting extra weight on the "budget" term leads to the worst portfolios selected, namely low-revenue and high-risk, compared to all the others. Additional insight is delivered by an average solution time per hyperparameter combination. It is clear from the bottom-right plot that low quality portfolios are selected faster with the largest average solution time being demonstrated by the group of fine-tuned parameters that also seem to deliver best solutions in terms of return and risk of selected portfolios.

The main takeaway from this setting is that potentially inclusion of the "budget constraint" into the cost function prevents the GNN-solver from selecting low-risk and high-revenue portfolios. The reason for this behavior is unclear. It can be hypothesized that the "budget" constraint is intrinsically discrete and loses its meaning when associated variables become continuous, as required by the GNN-solver. However, testing of this hypothesis is left for future research. Nevertheless, this results led to the testing of the GNN-solver in the experimental setting 3 where a modified cost function was used without "budget" term (see Equation (6.4)).

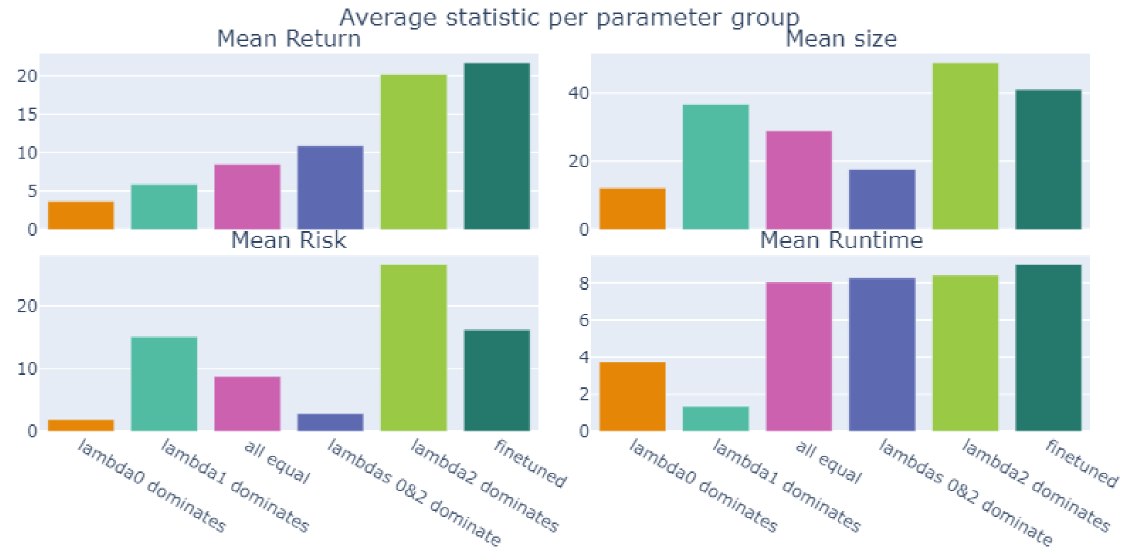


Figure 6.8: Average statistic per hyperparameter group.

6.5.3 Experimental Setup 3

In this experimental setup, the portfolio is selected based on the cost function as in Equation (6.4) with $\lambda = 1$.

Figures 6.9 and 6.10 show that the quality of the selected portfolios considerably improved compared to the ones derived in the Experimental Setting 1 (Figures 6.5 and 6.6). Again, portfolios selected by the GNN-solver are plotted against 100 randomly selected portfolios of the same size. Almost all of the selected portfolios have higher than random returns with low risk. Thus, it can be concluded that selecting portfolios without the "budget" constraint led to better quality solutions. The downside of this setup however is that it is impossible to directly control the number of assets in the portfolio selected, which can be important in some cases when requirements to the portfolio's diversity exist.

Nevertheless, even in the absence of the "budget" constraint, some insights into the optimal size of the portfolio can still be gained. Figure 6.11 shows that the sizes of the selected portfolios seem to be distributed around some mean value that depends on the size of the set of assets to select from (N). As will be demonstrated bellow, this size also depends on λ - weight put on the return part of the cost function. Such distribution over derived portfolio sizes suggests the existence of the optimal range for the portfolio size for the particular set of assets and respective risk-return weighting. Thus, for example, requiring a portfolio of size 50 for the current asset set will deliver sub-optimal portfolios for a 1:1 weighting of risk and return in the cost function.

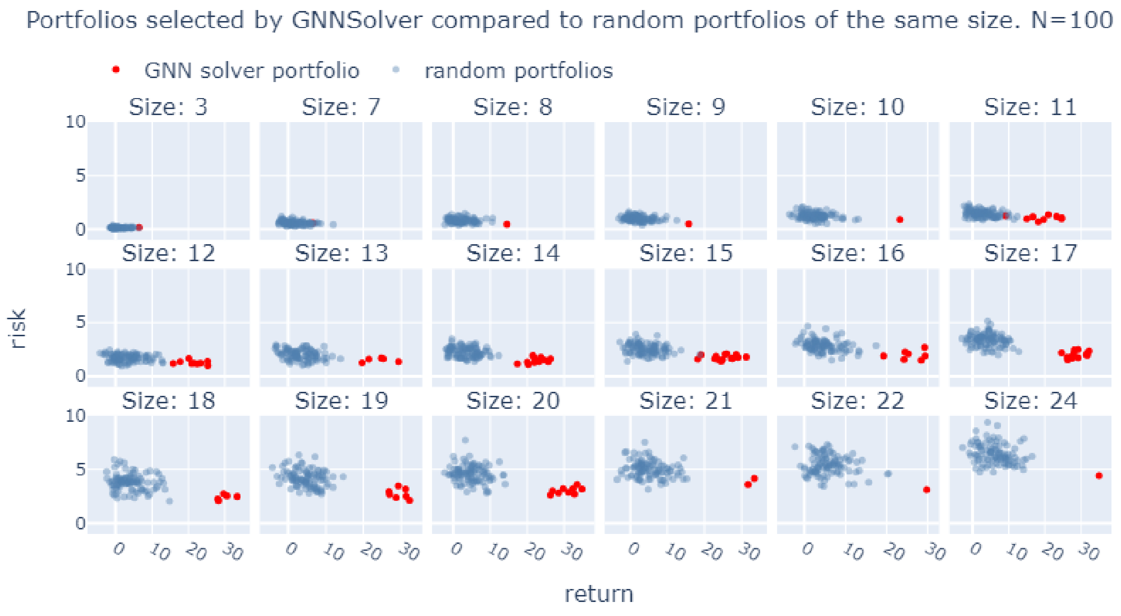


Figure 6.9: Portfolios selected by the GNNSolver compared to random portfolios of the same size. $N = 100$.

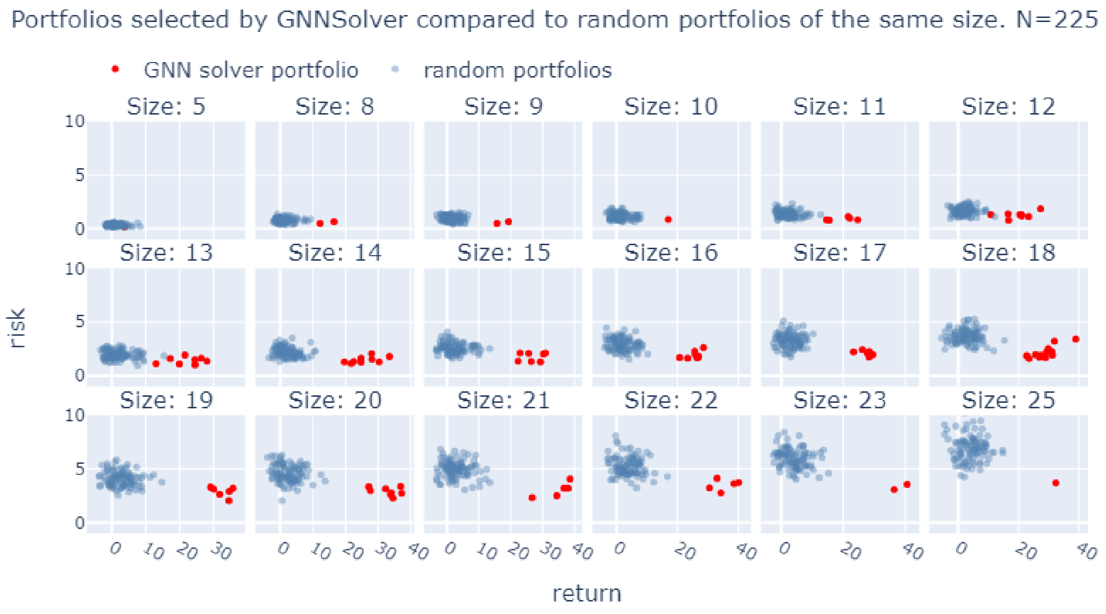


Figure 6.10: Portfolios selected by the GNNSolver compared to random portfolios of the same size. $N = 225$

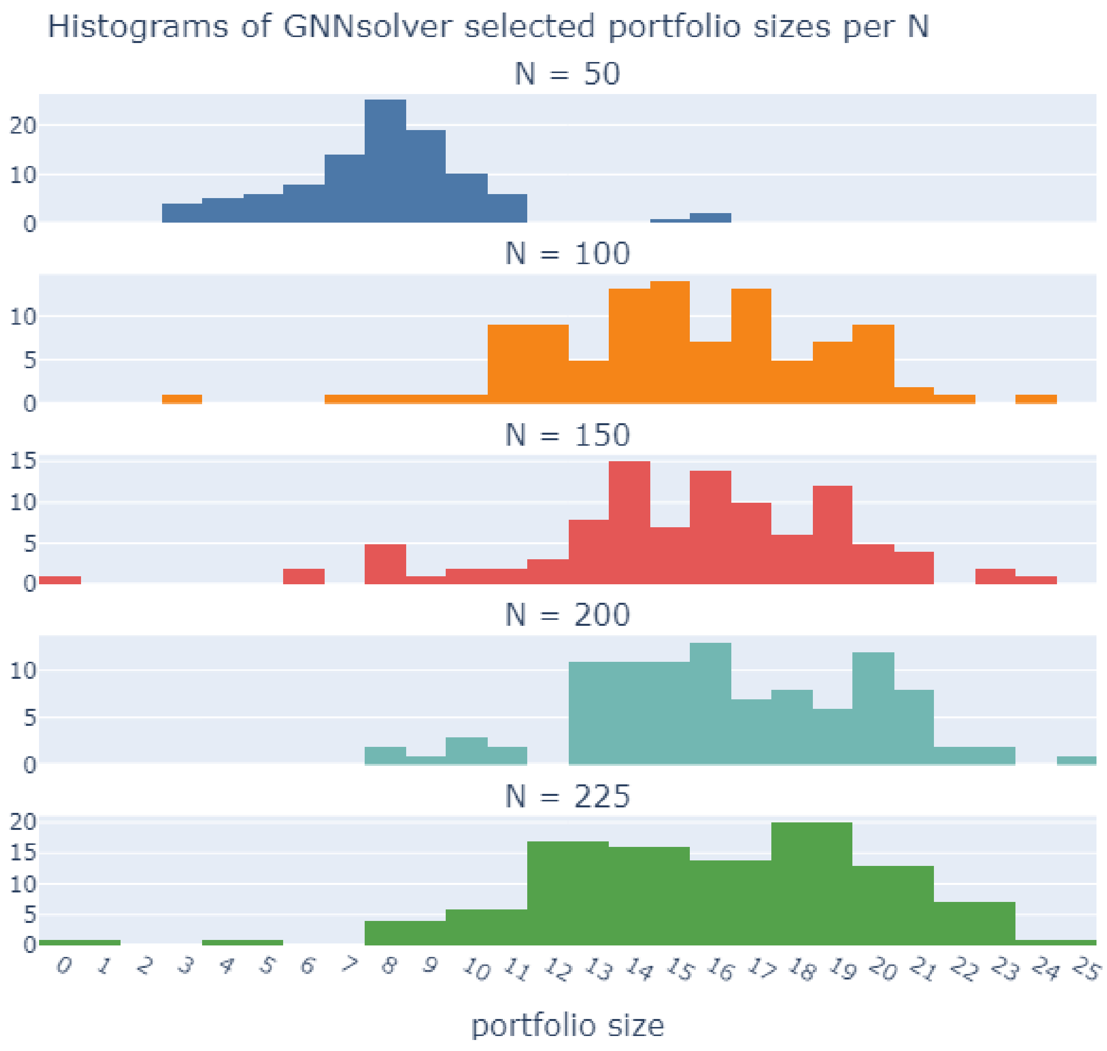


Figure 6.11: Histogram of GNNsolver selected portfolios sizes per N.

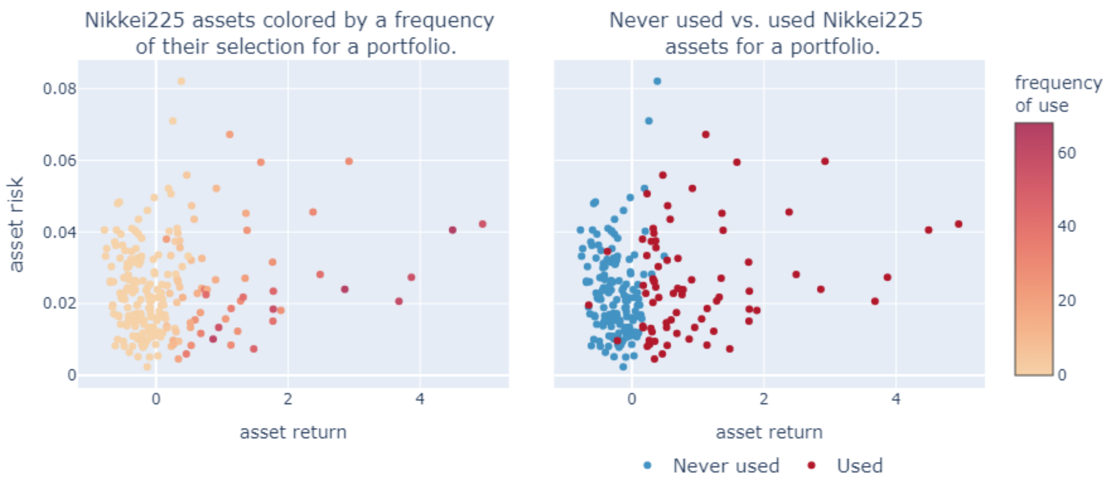


Figure 6.12: Use of Nikkei225 assets for portfolio selection by the GNN-solver.

Another useful insight that this experimental setting delivered is that its results can be used for the preselection of the available assets.

Figure 6.12, the right subplot, shows that there is a clear split between the assets that were never selected in 100 solution runs for a portfolio and those that were selected. The left subplot color highlights the relative frequency of an asset selection. As to be expected the mostly used assets are those that are plotted in the left bottom part of the scatter plot as high revenue- low risk assets. It should be noted that occasionally low-return and high-risk assets can still be beneficial for the optimal portfolio due to their negative correlations with other assets, nevertheless typically on average high-revenue and low-risk assets are more often preferred.

It should also be noted that the preferableness of an asset for a portfolio is only slightly influenced by the size of the set of assets to select from. While the actual frequency of the selection of a particular asset might decrease as the size of the asset set grows, the distribution of those frequencies over the same assets stays similar (see Figure 6.13).

Based on these frequency distributions, a pre-selection of the assets can be conducted before the actual optimization procedure. Working with the reduced sets will allow to reduce runtime for many existing "off-the-shelf" algorithms and also potentially deliver subsets small enough to be solved by the "pure" quantum annealing algorithm.

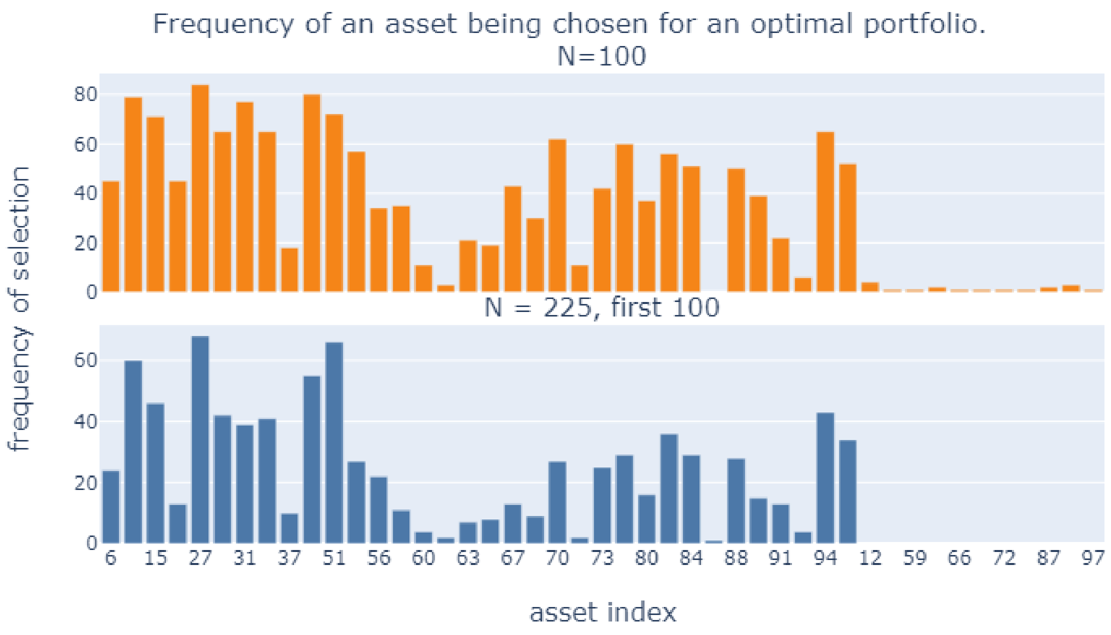


Figure 6.13: Comparative selection frequency of Nikkei225 assets per N.

6.5.4 Experimental Setup 4

Finally, the effect of assigning different weights (λ) on the return term of the cost function (Equation (6.4)) was investigated. For a set of weights $\lambda \in \{0.1, 0.5, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ 100 solution runs were performed, and 100 portfolios were selected by the GNN-solver.

The cost function used (Equation 6.4) does not explicitly impose a portfolio size. However the portfolio size is indirectly influenced by weighting of the return term. Figure 6.14 shows that increasing λ shifts the mean of the portfolio size to the right. The scale of the portfolio size distribution seems to grow with the number of assets to select from.

Solutions obtained by changing the weight on the return part of the cost function were used to construct the Pareto front, which is a set of all points that have the property of being Pareto optimal. "A point $x = (x_1, x_2, \dots, x_n)$ is said to be a Pareto optimal point when there exists no point $x^{'} = (x^{'}_1, x^{'}_2, \dots, x^{'}_n)$ which would have the property that on passing from x to $x^{'}$ the functions $f_1(x_1, \dots, x_n)$, $f_2(x_1, \dots, x_n)$, $\dots$ $f_N(x_1, \dots, x_n)$ do not decrease, and at least one of them effectively increases." (57). To construct the Pareto front, for each $\lambda \in \{0.1, 0.5, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ 5 solutions with the lowest obtained cost function value were selected. These solutions are compared against a random Pareto front and the Pareto front derived by Gurobi commercial solver (58). To derive random Pareto front the following procedure was followed: for each portfolio size selected

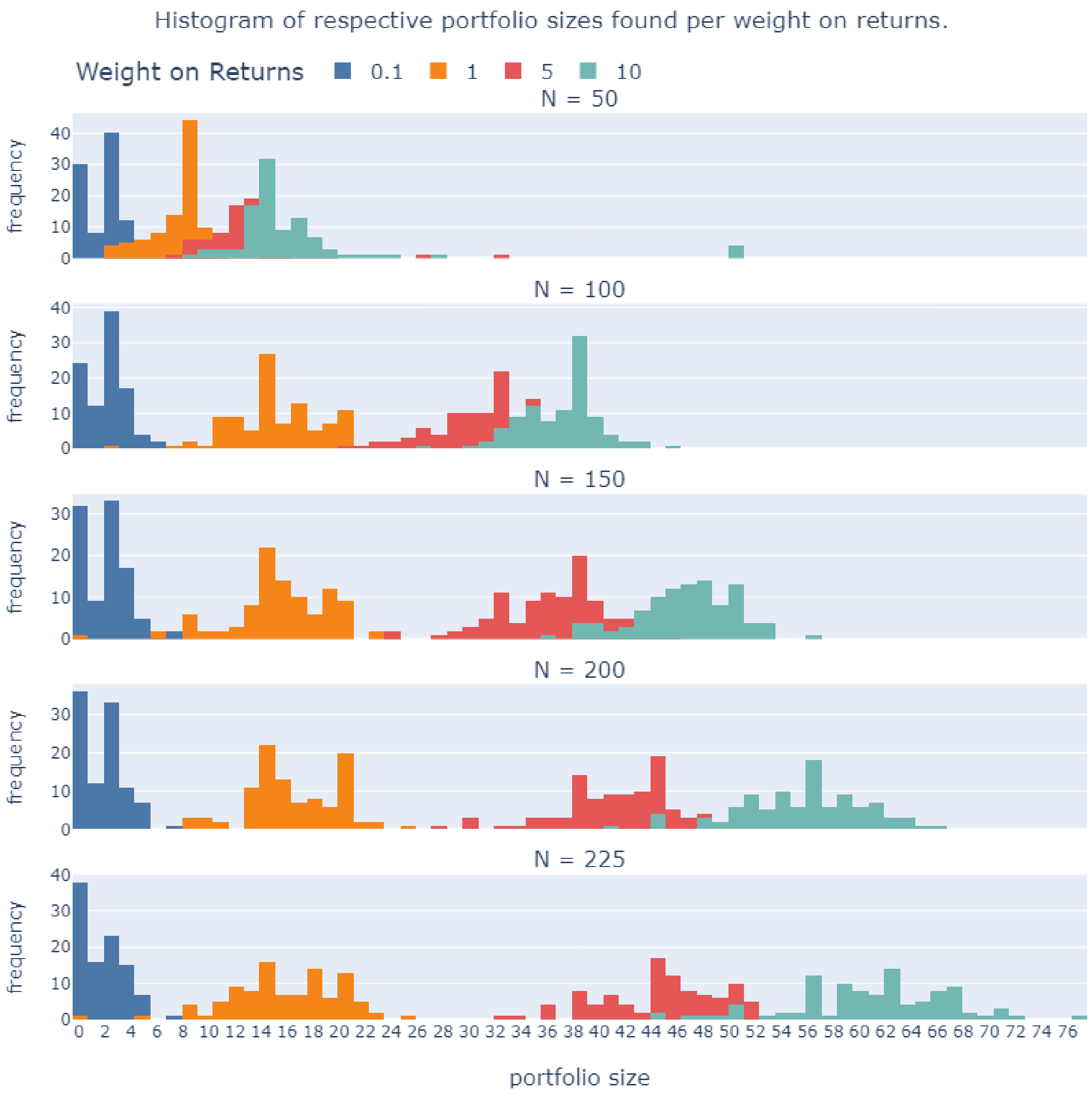


Figure 6.14: Histogram of respective portfolio sizes found per weight on returns.

by Gurobi solver 100 random portfolios were selected and an intermediate set of Pareto optimal portfolios per portfolio size was determined. Finally, a joint random Pareto front was selected out of the intermediate set of portfolios.

Figure 6.15 shows a comparison of those Pareto fronts. The GNN-solver Pareto front is obviously better than random and is very close to the Pareto front found by Gurobi solver, which proves that the GNN-solver is able to identify high quality portfolios based on the cost function defined in Equation 6.4 without a constraint on the portfolio size.

Figures 6.17 and 6.18 show that when compared by size GNN selected portfolios are very close by total portfolio risk and return to the portfolios selected by Gurobi.

Runtime Figure 6.16 shows that the average solution time per instance is below 2 seconds for all problem instances. Such run times easily allow for multiple solution runs to counterweight dependence of the solver on initialization.

The runtime per Pareto front is approximately defined as $t_{Pareto} = \tilde{t}_{solution} * k * r$, where $\tilde{t}_{solution}$ is an average solution time per instance, k is a number of points in the Pareto front (number of different λ's) and r is a number of solution runs per instance and λ. Since as was already mentioned above, GNN-solver is dependent on initialization, multiple runs are always required. Moreover as Figure 6.19 shows, the quality of a Pareto front depends on the number of runs r, especially for the larger sets of assets. The necessity to run multiple runs and associated runtime costs work to the detriment of the GNN-solver compared to Gurobi which is able to arrive at the Pareto solution in one solution run. Figure 6.20 shows the comparison of respective run-times per Pareto front. The mean solution time $\tilde{t}_{solution}$ here is averaged over all runs and λ's per N. The solution time per Pareto front seem to scale with N similar to Gurobi solution time especially for deriving Pareto fronts of a comparable quality (100 runs per instance).

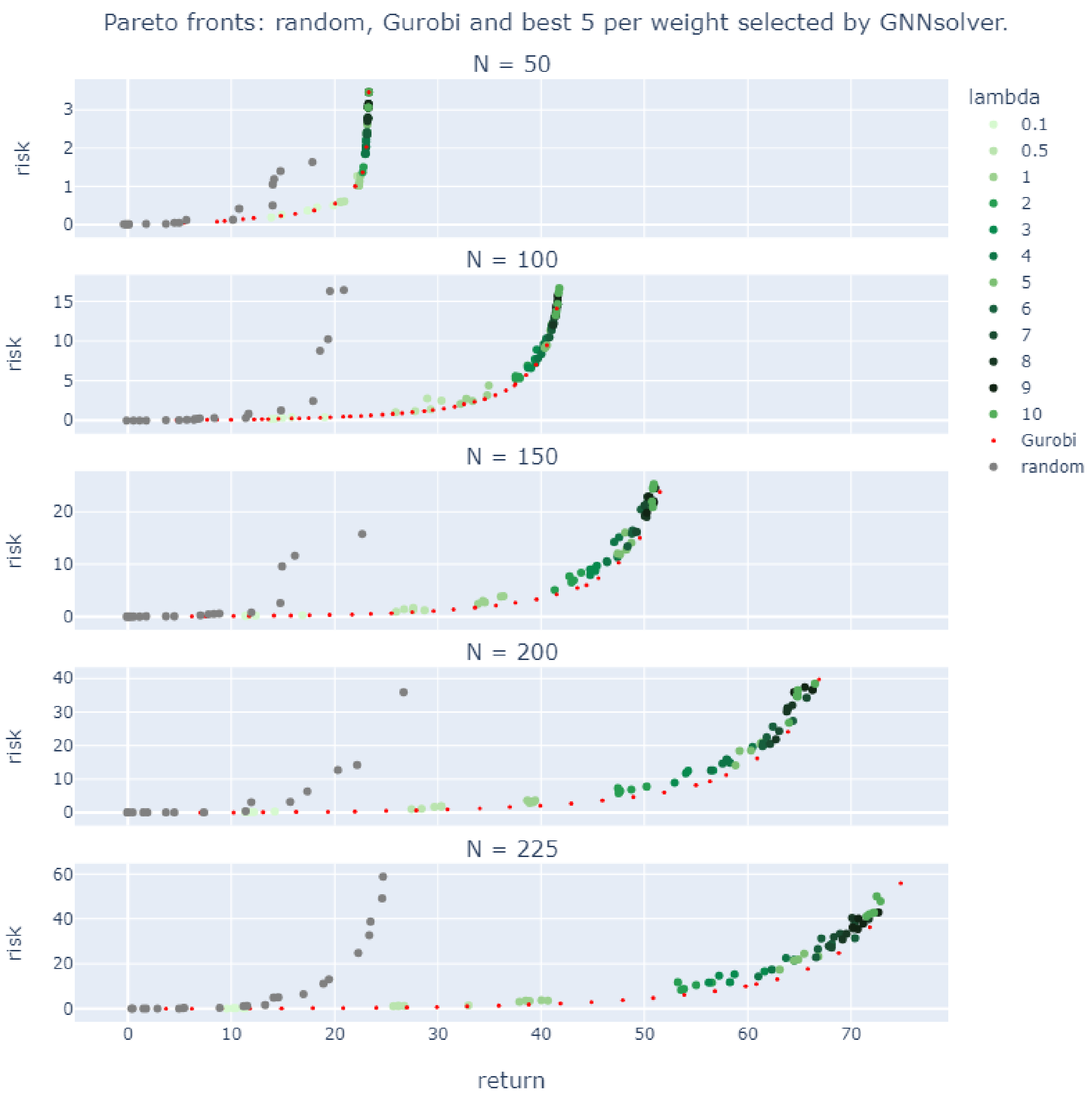


Figure 6.15: Pareto fronts: random, Gurobi, and best 5 per weight selected by GNNsolver.

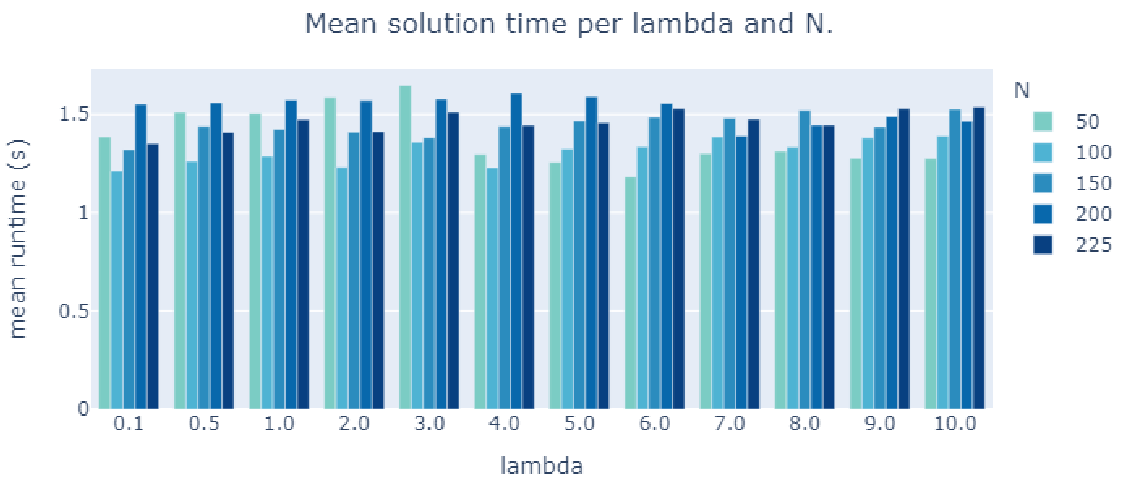


Figure 6.16: Mean GNN-solver solution time per λ and N.

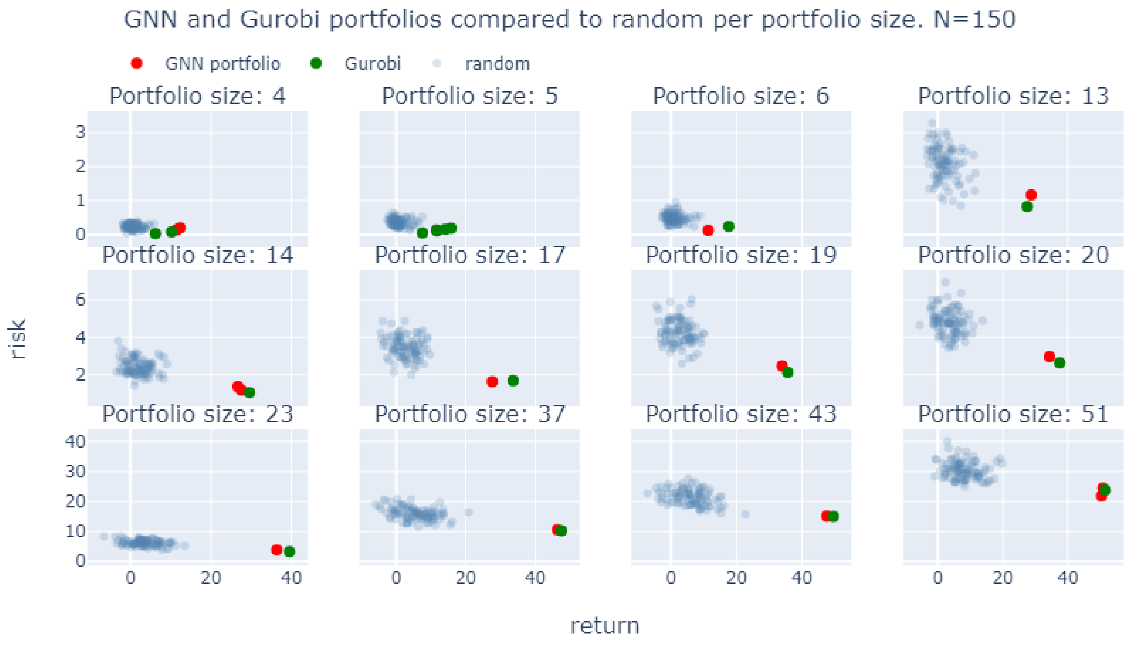


Figure 6.17: "Portfolios selected by GNN and Gurobi compared to random portfolios per portfolio size. $N = 150$"

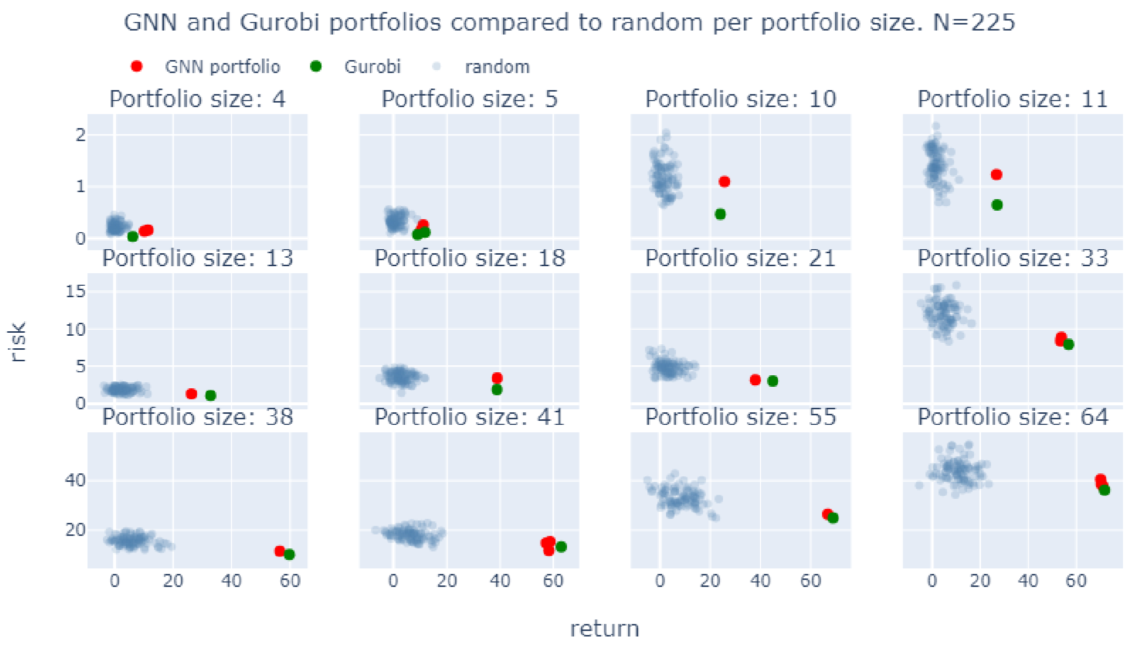


Figure 6.18: "Portfolios selected by GNN and Gurobi compared to random portfolios per portfolio size. $N = 225$"

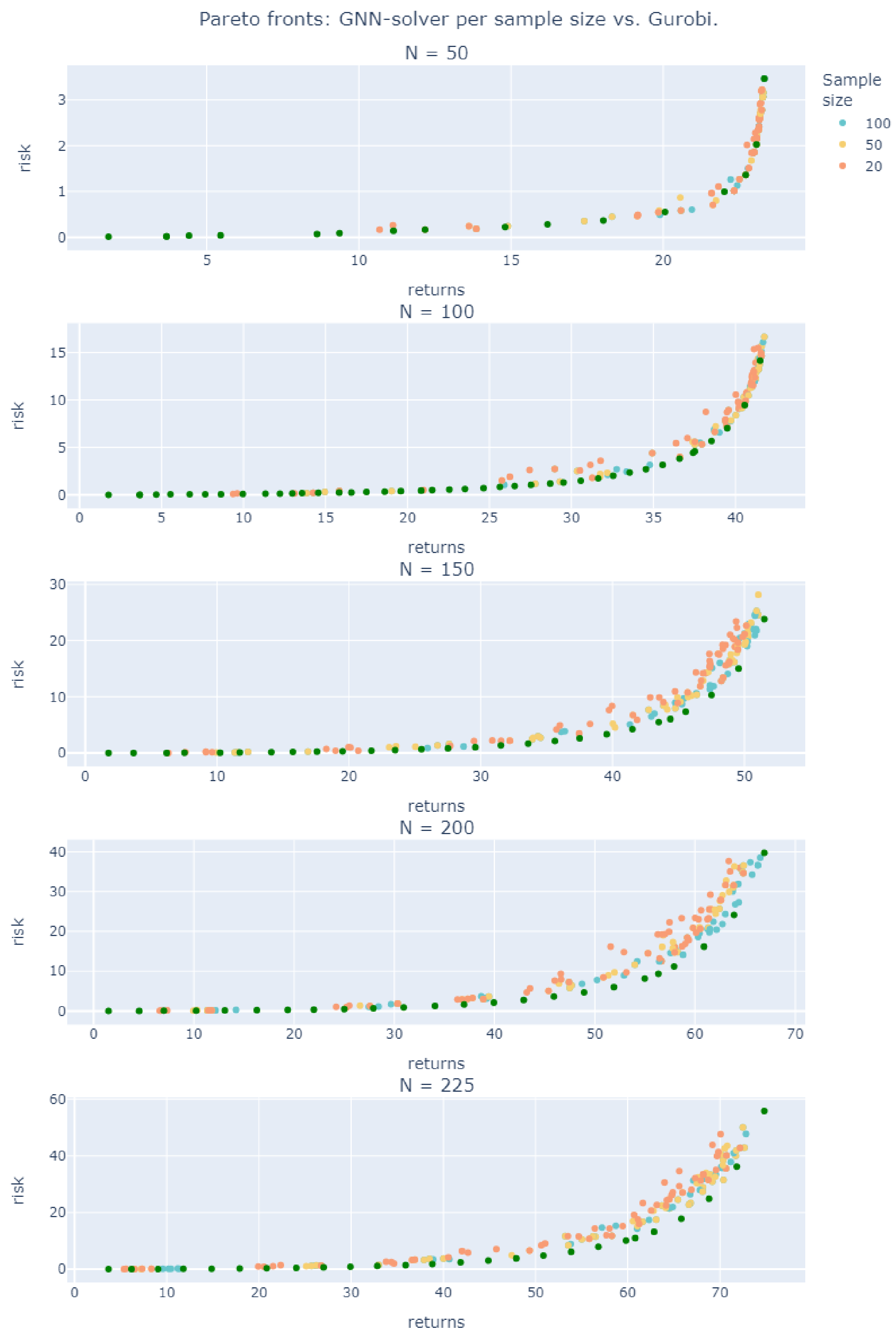


Figure 6.19: Pareto front per number of runs of GNN-solver per instance/lambda combination compared to Gurobi Pareto front.

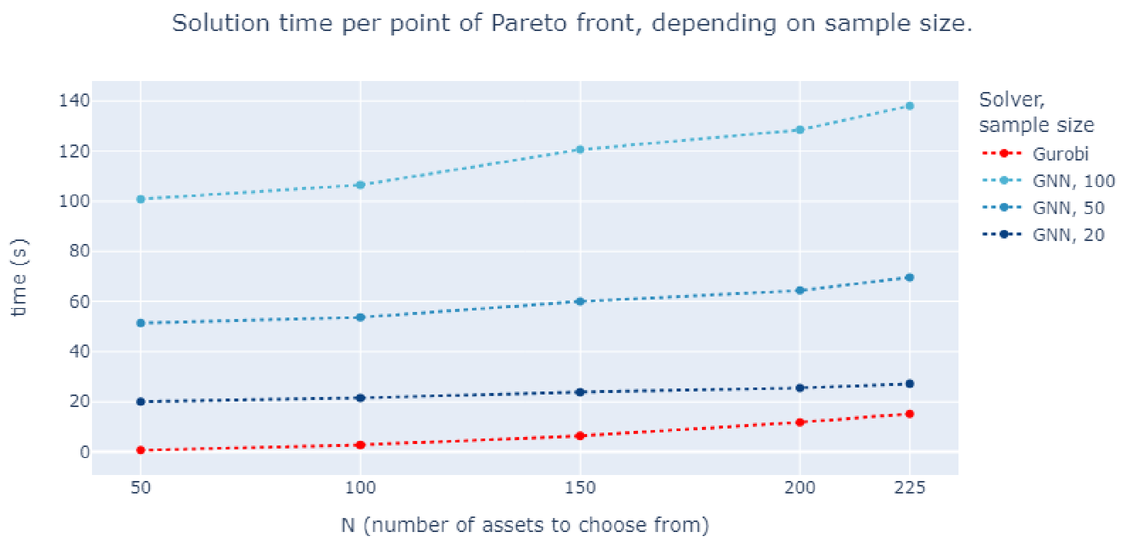


Figure 6.20: Solution time per point of Pareto front.

7

Conclusion and discussion

In this work, the application of the GNN-solver to several problems in QUBO formulation was studied to answer the question to which extend Graph Neural Networks can be leveraged to solve QUBO's. Specifically, it was investigated how the growing graph connectivity might influence the GNN-solver performance and to which extend the discovered negative effects can be moderated by different GNN architectures. Also performance of the GNN-solver was evaluated on the real-life use-case, namely, the *portfolio optimization problem* in its QUBO formulation.

It was found that the basic GNN-solver architecture proposed in (11) when applied to the MIS problem can only handle sparse graphs. The increasing degree of the graph especially combined with its regularity seems to pose an obstacle for a GCN-based architecture. To gain further insights into the potential causes of this problem, several modifications of gradient descent were applied to Max-Cut and MIS problems. The obtained results indeed confirmed as was previously pointed out in multiple researches that the gradient information obtained on discrete problems is not useful and usually cannot guide a solution process. However, this seems to depend on the problem at hand: the Max-Cut problem instances were susceptible to gradient descent methods while MIS problem instances were not. The exact reason for this different behavior per problem type was not found and is left for future research.

Further, it was demonstrated that addition of the graph information obtained through the graph convolutional layers can contribute positively to the solution process, compared to the gradient descent methods. However, the basic GCN-solver did not find solutions for non-sparse MIS instances. The potential cause for this is that learning useful node representations is also challenging when loss function gradients are not useful. This hypothesis can be investigated further in the future research by more detailed investigation of learned node embeddings and their discriminatory power, as was done in (7). Here, the following modifications to the GNN-solver were added. Experimental results indicate that these modified architectures of the GNN-solver are able to considerably broaden the scope of MIS graph instances that can be efficiently solved in terms of their increasing density:

Embedding transfer. Learning graph representations on the Max-Cut problem and transferring them to the MIS instances enhanced the ability of the GCN solver to solve denser graph problems.

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

GraphSAGE layers. Replacing GCN layers with GraphSAGE layers showed the best results in broadening the scope of the MIS instances that can be solved in terms of increasing graph degree. However, it was concluded that while offering opportunities to solve denser MIS instances, GraphSAGE-based solvers are still impeded by the growing degree of the graph. This can be due to the limitations of the GNN architectures, as pointed out in previous research and discussed in Section 2. Investigating other graph neural network architectures with enhanced expressivity is a potentially promising topic for further research.

In addition to the numerical experiments the GNN-solver was applied to the real-life case of selecting efficient portfolio from Nikkie225 index. It was shown that the performance of the GNN-solver depends on the particular choice of QUBO cost function at hand. The results suggest that some constraints incorporated into the cost function can be misleading for the GNN-solver and impede the effective portfolio selection. In general, experiments conducted suggest that the GNN-solver has difficulty respecting hard constraints in the portfolio optimization case, which is not surprising since this inability has been already pointed in previous research (21).

When applied to the portfolio optimization problem formulated without "budget" constraint, the GNN-solver demonstrated reasonably good performance with the best solutions found placed on the Pareto front derived by the Gurobi commercial solver. Interestingly, portfolio selection case in its MIS formulation represents a fully connected graph, where adjacency matrix is represented by risk matrix. However, high graph connectivity in this case did not impede solution process, potentially because the adjacency matrix A which is discrete in a classical MIS problem, is represented by a continuous risk matrix in the portfolio selection case. This might suggest that when relaxation is applied to the CO problem for the sake of obtaining a continuous loss function, relaxation of the discrete variables might not be enough. Indicating the real-life cases for which adjacency matrix relaxation can also be meaningful might be an interesting topic for future research as well.

It was also demonstrated how the application of the GNN-solver to the portfolio selection problem can still be useful by delivering helpful insights into the problem even when it cannot produce feasible solutions to the problem specification. Namely, the GNN-solver can be used as a *clever sampler* of the solutions to indicate for example which portfolio sizes are optimal for the specified ratio between risk and return in the required portfolio. It can also be used for preselection of the potentially useful assets from the broader set of the assets given by discarding obviously undesirable items.

In general it can be concluded that the GNN-solver in its modified version presented in this research is a potentially useful tool for solving CO problems. However, it is not an universal solver and its applicability and limitations should be decided on case-by-case basis offering a lot of potential avenues for future research.

References

[1] Frank Phillipson. **Searching for Optimization**. Inaugural Lecture, 2022. 1

[2] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. **Machine learning for combinatorial optimization: a methodological tour d'horizon**. *European Journal of Operational Research*, **290**(2):405–421, 2021. 1

[3] James Kotary, Ferdinando Fioretto, Pascal Van Hentenryck, and Bryan Wilder. **End-to-end constrained optimization learning: A survey**. *arXiv preprint arXiv:2103.16378*, 2021. 1, 2, 4

[4] Natalia Vesselinova, Rebecca Steinert, Daniel F Perez-Ramirez, and Magnus Boman. **Learning combinatorial optimization on graphs: A survey with applications to networking**. *IEEE Access*, **8**:120388–120416, 2020. 2, 4

[5] Quentin Cappart, Didier Chételat, Elias B Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. **Combinatorial optimization and reasoning with graph neural networks**. *Journal of Machine Learning Research*, **24**(130):1–61, 2023. 2, 5

[6] Marcelo Prates, Pedro HC Avelar, Henrique Lemos, Luis C Lamb, and Moshe Y Vardi. **Learning to solve np-complete problems: A graph neural network for decision tsp**. In *Proceedings of the AAAI Conference on Artificial Intelligence*, **33**, pages 4731–4738, 2019. 2

[7] Henrique Lemos, Marcelo Prates, Pedro Avelar, and Luis Lamb. **Graph colouring meets deep learning: Effective graph neural network models for combinatorial problems**. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 879–885. IEEE, 2019. 2, 6, 62

[8] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. **Combinatorial optimization with graph convolutional networks and guided tree search**. *Advances in neural information processing systems*, **31**, 2018. 2

[9] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. **Learning combinatorial optimization algorithms over graphs**. *Advances in neural information processing systems*, **30**, 2017. 2

[10] Wouter Kool, Herke Van Hoof, and Max Welling. **Attention, learn to solve routing problems!** *arXiv preprint arXiv:1803.08475*, 2018. 2

[11] Martin JA Schuetz, J Kyle Brubaker, and Helmut G Katzgraber. **Combinatorial optimization with physics-inspired graph neural networks**. *Nature Machine Intelligence*, **4**(4):367–377, 2022. 3, 7, 16, 17, 18, 22, 25, 26, 30, 31, 61

[12] Priya Donti, Brandon Amos, and J Zico Kolter. **Task-based end-to-end model learning in stochastic optimization**. *Advances in neural information processing systems*, **30**, 2017. 4

[13] Brandon Amos and J Zico Kolter. **Optnet: Differentiable optimization as a layer in neural networks**. In *International Conference on Machine Learning*, pages 136–145. PMLR, 2017. 4

[14] Jayanta Mandi and Tias Guns. **Interior point solving for lp-based prediction+ optimisation**. *Advances in Neural Information Processing Systems*, **33**:7272–7282, 2020. 4

[15] Marin Vlastelica, Anselm Paulus, Vít Musil, Georg Martius, and Michal Rolínek. **Differentiation of blackbox combinatorial solvers**. *arXiv preprint arXiv:1912.02175*, 2019. 4

[16] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. **Weisfeiler and leman go neural: Higher-order graph neural networks**. In *Proceedings of the AAAI conference on artificial intelligence*, **33**, pages 4602–4609, 2019. 5

[17] Vikas Garg, Stefanie Jegelka, and Tommi Jaakkola. **Generalization and representational limits of graph neural networks**. In *International Conference on Machine Learning*, pages 3419–3430. PMLR, 2020. 5

[18] Nikolaos Karalias and Andreas Loukas. **Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs**. *Advances in Neural Information Processing Systems*, **33**:6659–6672, 2020. 5

[19] Martin JA Schuetz, J Kyle Brubaker, Zhihuai Zhu, and Helmut G Katzgraber. **Graph coloring with physics-inspired graph neural networks**. *Physical Review Research*, **4**(4):043131, 2022. 6

[20] Weichi Yao, Afonso S Bandeira, and Soledad Villar. **Experimental performance of graph neural networks on random instances of max-cut**. In *Wavelets and Sparsity XVIII*, **11138**, pages 242–251. SPIE, 2019. 6

[21] Hao Xu, Ka Hei Hui, Chi-Wing Fu, and Hao Zhang. **TilinGNN: learning to tile with self-supervised graph neural network**. *arXiv preprint arXiv:2007.02278*, 2020. 7, 62

[22] Abraham P Punnen. *The Quadratic Unconstrained Binary Optimization Problem.* Springer, 2022. 9

[23] Haskell B Curry. **The method of steepest descent for non-linear minimization problems**. *Quarterly of Applied Mathematics*, **2**(3):258–261, 1944. 10, 13

[24] Xue Li and Yuanzhi Cheng. **Understanding the message passing in graph neural networks via power iteration clustering**. *Neural Networks*, **140**:130–135, 2021. 10

[25] Jian Tang and Renjie Liao. *Graph Neural Networks for Node Classification*, pages 41–61. Springer Nature Singapore, Singapore, 2022. 10, 12

[26] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. **Message passing neural networks**. *Machine learning meets quantum physics*, pages 199–214, 2020. 11

[27] Thomas N Kipf and Max Welling. **Semi-supervised classification with graph convolutional networks**. *arXiv preprint arXiv:1609.02907*, 2016. 12

[28] Qimai Li, Zhichao Han, and Xiao-Ming Wu. **Deeper insights into graph convolutional networks for semi-supervised learning**. In *Proceedings of the AAAI conference on artificial intelligence*, **32**, 2018. 12

[29] Will Hamilton, Zhitao Ying, and Jure Leskovec. **Inductive representation learning on large graphs**. *Advances in neural information processing systems*, **30**, 2017. 13

[30] Charu C Aggarwal et al. **Neural networks and deep learning**. *Springer*, **10**(978):3, 2018. 13

[31] Sebastian Ruder. **An overview of gradient descent optimization algorithms**. *arXiv preprint arXiv:1609.04747*, 2016. 13

[32] Diederik P Kingma and Jimmy Ba. **Adam: A method for stochastic optimization**. *arXiv preprint arXiv:1412.6980*, 2014. 14

[33] Martin Riedmiller and Heinrich Braun. **A direct adaptive method for faster backpropagation learning: The RPROP algorithm**. In *IEEE international conference on neural networks*, pages 586–591. IEEE, 1993. 15

[34] Fred Glover, Gary Kochenberger, and Yu Du. **A tutorial on formulating and using QUBO models**. *arXiv preprint arXiv:1811.11538*, 2018. 16, 17, 21

[35] Mohamad Mahdi Mohades and Mohammad Hossein Kahaei. **An Efficient Riemannian Gradient Based Algorithm for Max-Cut Problems**. *IEEE Transactions on Circuits and Systems II: Express Briefs*, **69**(3):1882–1886, 2021. 17

[36] Richard M. Karp. *Reducibility Among Combinatorial Problems*, pages 219–241. Springer Berlin Heidelberg, 2010. 17

[37] MX Goemans. **Improved approximation algorithms for maximum cut and satisability problems using semidenite programming**. *Journal of Assoc. Comput. Mach.*, **42**:330–343, 1995. 17

[38] Rangaswami Balakrishnan and Kanna Ranganathan. *A textbook of graph theory*. Springer Science & Business Media, 2012. 17

[39] Md Saidur Rahman et al. *Basic graph theory*, **9**. Springer, 2017. 17

[40] Panos M Pardalos and Jue Xue. **The maximum clique problem**. *Journal of global Optimization*, **4**:301–328, 1994. 17

[41] Michael R Garey and David S Johnson. **"strong"np-completeness results: Motivation, examples, and implications**. *Journal of the ACM (JACM)*, **25**(3):499–508, 1978. 18

[42] Aric Hagberg, Pieter Swart, and Daniel S Chult. **Exploring network structure, dynamics, and function using NetworkX**. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008. 18

[43] NetworkX Developers. **Networkx. Approximations and Heuristics**. 19, 20

[44] Magnús Halldórsson and Jaikumar Radhakrishnan. **Greed is good: Approximating independent sets in sparse and bounded-degree graphs**. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 439–448, 1994. 19

[45] Ravi Boppana and Magnús M Halldórsson. **Approximating maximum independent sets by excluding subgraphs**. *BIT Numerical Mathematics*, **32**(2):180–196, 1992. 20

[46] Martin JA Schuetz, J Kyle Brubaker, Zhihuai Zhu, and Helmut G Katzgraber. **Graph coloring with physics-inspired graph neural networks**. *Physical Review Research*, **4**(4):043131, 2022. 24

[47] Joseph Bowles, Alexandre Dauphin, Patrick Huembeli, José Martinez, and Antonio Acín. **Quadratic Unconstrained Binary Optimization via Quantum-Inspired Annealing**. *Physical Review Applied*, **18**(3):034016, 2022. 27, 28

[48] Saurabh Agarwal. *Portfolio selection using multi-objective optimisation.* Springer, 2017. 38

[49] Maciej J Capiński and Ekkehard Kopp. *Portfolio theory and risk management*, **5**. Cambridge University Press, 2014. 38, 39

[50] Harry M Markowits. **Portfolio selection**. *Journal of finance*, **7**(1):71–91, 1952. 38

[51] Yuanyuan Zhang, Xiang Li, and Sini Guo. **Portfolio selection problems with Markowitz's mean–variance framework: a review of literature**. *Fuzzy Optimization and Decision Making*, **17**:125–158, 2018. 38

[52] Frank Phillipson and Harshil Singh Bhatia. **Portfolio optimisation using the d-wave quantum annealer**. In *International Conference on Computational Science*, pages 45–59. Springer, 2021. 39, 40, 42, 43, 45

[53] **How Quantum Annealing Works in D-Wave QPUs**. `https://docs.dwavesys.com/docs/latest/c_gs_2.html#getting-started-qa`. Accessed: 2024-02-07. 41

[54] Salvador E Venegas-Andraca, William Cruz-Santos, Catherine McGeoch, and Marco Lanzagorta. **A cross-disciplinary introduction to quantum annealing-based algorithms**. *Contemporary Physics*, **59**(2):174–197, 2018. 41

[55] **Solver Docs. Using Leap's Hybrid Solvers**. `https://docs.dwavesys.com/docs/latest/doc_leap_hybrid.html`. Accessed: 2024-02-07. 42

[56] Sotiris B Kotsiantis, Dimitris Kanellopoulos, and Panagiotis E Pintelas. **Data preprocessing for supervised leaning**. *International journal of computer science*, **1**(2):111–117, 2006. 45

[57] Ragnar Frisch and Ragnar Frisch. **Simultaneous Search for the Extrema of Several Functions. Pareto Optimality**. *Maxima and Minima: Theory and Economic Applications*, pages 48–59, 1966. 53

[58] **Gurobi optimizer reference manual.** `https://www.gurobi.com/documentation/11.0/refman/index.html`. Accessed: 2024-02-12. 53