POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Graph Neural Networks and Reinforcement Learning for Minor Embedding in Quantum Annealing

Laurea Magistrale in High performance Computing Engineering

Author: Giorgio Miani

Advisor: Prof. Maurizio Ferrari Dacrema

Co-advisor: Riccardo Nembrini

Academic year: 2023-2024

1. Introduction

Quantum Annealers (QA) [4] are a paradigm of Quantum Computers designed to solve combinatorial optimization problems by leveraging quantum mechanical laws, such as quantum tunnelling and entanglement. Quantum Annealers are a promising technology with the potential of offering a relevant speed-up w.r.t. classical computers when solving NP-hard combinatorial problems. One of the main challenges when solving a problem on a QA, is the process of mapping the problem written in the QUBO formulation [2] to the topology of the QA. This process is called Minor Embedding [1]. Due to the variable structure of the problem to be solved and the rigid and sparse structure of the Quantum Annealers, finding a good mapping can become computationally expensive and it can reduce the advantage obtained by using a Quantum Computer. Traditional embedding methods sometimes can be suboptimal or inefficient for complex QUBO problems.
Our goal is to address this challenge by leveraging advancements in the rapidly evolving field of Machine Learning on graphs. This field blends graph theory with machine learning algorithms, for example it includes Graph Neural Networks (GNNs), a family of deep learning architectures designed to work on graphs. Our objective is to develop an innovative approach based on this field capable of finding valid Minor Embeddings for a generic problem on a specific Quantum Annealer. We can represent the QUBO problem as a graph, called **G**, where each node is a variable and we can also represent the topology of a QA as a graph, called **H**, where each node is a qubit and each edge is a possible connection between two qubits.
We designed a PPO-based [5] Reinforcement Learning agent, called GReM (Graph-based Reinforcement learning for Minor embedding), that exploits Graph Neural Networks to iteratively map a node of **G** to a node of **H**, until the problem is embedded. The model is trained on a dataset composed of connected graphs with 3 to 11 nodes representing generic QUBO problems, ordered by increasing size and equally distributed according to their isomorphisms and their number of edges. The embeddings were performed on D-Wave Chimera and Zephyr topologies with up to 1152 and 2176 qubits, respectively. The quality of the results are compared with state-of-the-art heuristic MinorMiner's embeddings, given that, for the prob-

lems faced, this heuristic's solutions are near-optimal.

2. GReM Design

GReM is designed to work as a PPO-based Deep Reinforcement Learning algorithm. We developed two versions of it: one that incorporates a Curriculum Learning (CL) process and the other one without it. CL is a paradigm of Reinforcement Learning where the problem is presented to the model with increasing difficulty. The main components of the model are the performable actions, the state of the environment, the reward and the agent. For training and testing phases we used a dataset composed of a set of graphs with different sizes, designed to help the model in the learning process.

2.1. Action

The agent interacts with the environment by observing its state and choosing an action accordingly. At each step a node of **G** is selected in a cyclic order using a round robin scheduling. An action taken by the model consists of mapping the selected node of **G** to an available node in **H** . The node in **H** is chosen by sampling from a probability distribution computed by the agent. An Invalid Action Masking (IAM), that sets to zero the probability of choosing unavailable nodes, prevents the agent from choosing qubits which are already mapped to a variable.

2.2. State

The state of the environment is composed of a combination of two graphs: the problem graph, denoted as **G**, and the topology graph, denoted as **H**. When performing training, **G** is sampled from a dataset containing multiple graphs with varying sizes, ranging from 3 to a maximum size defined by an hyper-parameter, while **H** is fixed, with both its size and topology determined by hyper-parameters. Nodes in both graphs contain specific features.
Each node in **G** has three features:

- **degree**: Represents the degree of the node.
- **chain**: Defined as a vector with a length equal to the number of nodes in **H**; each element is **1** if the **G** node is mapped to the corresponding node in **H**, and **0** otherwise.
- **round robin**: A scalar value set to **0** if the corresponding node of **G** has to be mapped in the current step, **1** otherwise.

Each node in **H**, on the other hand, has a single feature corresponding to its degree.

2.3. Reward

After each action, the agent receives a negative reward which is fixed to **-0.1**. If all qubits in **H** are unavailable, but the embedding found is not valid, the reward is set to **-1**.

2.4. Agent

The agent, based on the PPO algorithm, is composed of an actor and a critic, both with similar architectures. Each consists of a GNN divided into two towers: one processing **G** and the other **H**. The **G** tower contains a single Graph Convolution Network (**GCN**) layer followed by a **global mean pooling** layer. The **H** tower includes three **GCN** layers, whose outputs are concatenated and passed through a **global mean pooling** layer. The outputs of both towers are then concatenated and passed through three dense layers with *tanh* activation functions. The critic outputs a single value corresponding to the estimate of the total reward we expect to receive, while the actor produces a number of values equal to the number of qubits, representing the probabilities for each qubit to be selected. IAM sets to zero the probability for qubits already mapped to a variable.

2.5. Training

To train our model, we use the PPO algorithm implemented by *CleanRL* [3]. We developed two versions of it, one designed to train with a CL process and the other one without it. Most of the hyper-parameters used are set using the default ones of *CleanRL*, but additional hyper-parameters have been introduced for CL.
We are going to discuss the common parts of the two alternatives first. For a fixed number of steps the model repeats the following procedure: the agent receives an observation containing the features of **G** and **H**, processes it, and produces a probability distribution used to sample the next action. The model performs an action by mapping the current node of **G** to a node of **H** and receives a reward and a new observation, which is processed in the next iteration. Then, after collecting a batch of steps, the model enters the roll-out phase, where the

agent's weights are updated. Once the roll-out phase is completed, the model repeats the whole process. While training, the model faces an increasing level of difficulty for the problems. It starts by embedding graphs with 3 nodes and the graphs' size increases until it reaches the maximum size of the dataset. The condition that has to be satisfied in order to switch to the next size depends on the implemented version.
In the CL version, the process continues to embed graphs with a fixed size until a convergence criterion is satisfied. Upon convergence, training proceeds to the next size and repeats the process again. Training stops when the model reaches convergence for graphs with maximum size. Convergence is based on the quota of **G** graphs embedded and a threshold for this quota is used to determine the model's convergence. Both the threshold and the learning rate (LR) decrease linearly during training and the LR becomes constant when reaching a fixed lower bound. Upon convergence, training resets the LR and the threshold to their initial values. Moreover, we noticed that for each graph size, reintroducing a small percentage of previously trained graphs helps to maintain the model's proficiency on smaller graphs even as it advances to larger ones. The percentage of smaller graphs reintroduced is fixed at the beginning of training.
Instead, in the basic implementation without CL, a fixed number of graphs for each size is selected and when the agent has computed an embedding for all graphs with a specific size, it moves to the next one. When the model has processed the whole dataset, the process repeats from the minimum size, with no reset of the training phase. This continues until the model has exhausted a given number of training steps. No condition is used to determine if the model has reached convergence.

2.6. Testing

During the testing phase, the trained model is loaded to compute embeddings for the testing graphs in the dataset. The testing graphs have the same sizes of the ones in the training set, but are never seen by the model during training. Moreover, the graphs are equally distributed according to their number of edges and their connectivity, similarly to the training set.

3. Results

In this section we are going to show and analyse the results obtained from the tests discussed in Section 2.6. First we are going to look at the results for the complete model described in section 2. Then we are going to discuss the key components of the model together with the motivation and the results that lead to using such features. All results shown, except for section 3.3.1, are obtained by collecting 10 experiments using different seeds. Moreover, the model and MinorMiner are set to find 10 embeddings for each problem graph in the testing dataset. The results of MinorMiner shown are obtained considering the smallest embedding found for each problem graph, while for our model it depends on the considered metric.

3.1. Metrics

The evaluation of the quality of the results for the different versions of the model is based on three main criteria:

Success Rate (SR) The success rate is determined by the number of problem graphs in the testing dataset successfully embedded by the model. A **G** graph is successfully embedded if all of the 10 embeddings produced for that graph are valid.

Qubit Utilization Ratio (QUR) Provided the success rate is sufficiently high, we compare the number of qubits used by our model with those used by MinorMiner, employing the following ratio:

$$\text{Qubit Utilization Ratio} = \frac{|E_{\text{RL}}| - |E_{\text{MM}}|}{|E_{\text{RL}}|} \quad (1)$$

where $|E_{\text{RL}}|$ and $|E_{\text{MM}}|$ represent the number of qubits of the smallest embedding found for a single problem graph in the test set by our model and MinorMiner, respectively. Considering the size of the problems we use to test our model, MinorMiner is highly efficient at finding good embeddings using a minimal number of qubits. This makes it an almost optimal reference point for comparing our results. An efficient embedding minimizes the use of qubits. Therefore, the closer $|E_{\text{RL}}|$ is to $|E_{\text{MM}}|$, the better the embedding quality, moving the ratio closer to zero.

Conversely, excessive qubit usage by the model will increase $|E_{\mathrm{RL}}|$ w.r.t. $|E_{\mathrm{MM}}|$, pushing the ratio towards 1.

Execution Time Ratio (ETR) Lastly, we examine the time required by the model to compute an embedding during testing. We compare the results with MinorMiner performance by computing the Execution Time Ratio as:

$$\text{Execution Time Ratio} = \frac{T_{RL}}{T_{MM}} \tag{2}$$

where T_{RL} is the time taken by the model to find the smallest embedding for a problem graph in the test set. On the other hand, T_{MM} is the minimum time taken by MinorMiner across all 10 embeddings found for the same problem graph. After detailing these metrics, we proceed to evaluate the quality of the results obtained by the model, as described in Section 2.

3.2. GReM

The primary objective is to develop a model capable of finding valid embeddings for a generic problem on a specific Quantum Annealer. In addition to ensuring embedding validity, we aim to minimize the number of qubits required and optimize the model's computational efficiency.
For the following experiments we have used default values for the *CleanRL* hyper-parameters, except for the number of epochs used to update the weights set to 10. The number of steps considered for each **G** size is $3M$, the threshold is set to $5K$ graphs and the learning rate lower bound is $1.0e^{-4}$. For the dataset we have chosen to use approximatively 1250 graphs per size. For each size, around 80% of the graphs are used for training only and the remaining ones for testing. Moreover, during training, for each size, 500 smaller graphs are resampled from the dataset. All results were collected from 10 experiments conducted on graphs with node counts ranging from 3 to 11, embedded within a Zephyr topology of 2176 qubits. For each of these experiments we train the same model using different seeds.
The results indicated that GReM's SR is **100%**. Having established this capability, we proceed to analyse the model's QUR. Figure 1 depicts the QUR for different **G** sizes derived from the same experiments used to evaluate the SR. The results

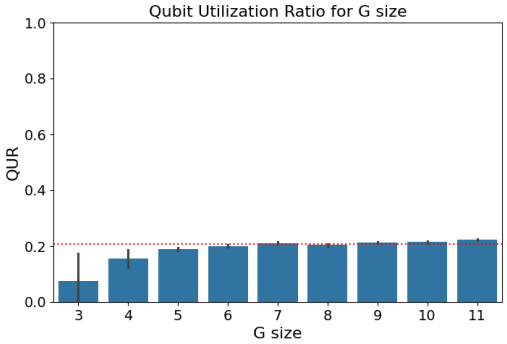

Figure 1: Average QUR across 10 experiments for different **G** sizes. The X-axis represents the **G** size, and the Y-axis shows the QUR.

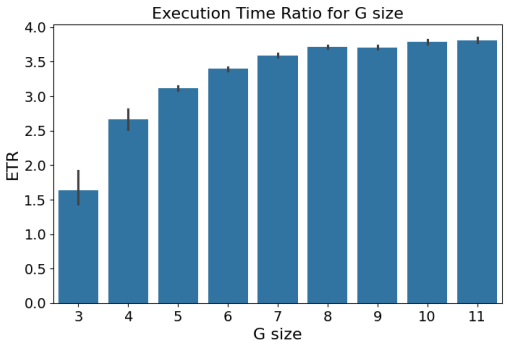

Figure 2: Average ETR across 10 experiments for different **G** sizes. The X-axis represents the **G** size, and the Y-axis shows the ETR.

demonstrate that GReM achieves embeddings comparable to those generated by MinorMiner. The average QUR across all sizes indicates that MinorMiner uses, on average, **20.89%** fewer qubits than the model. The best QUR obtained for a single experiment corresponds to **14.37%**. Additionally, by analysing the experiments we noticed that the model tends to find better embeddings for larger graphs when it performs well on smaller ones. Lastly, we evaluate the ETR obtained by the model. Figure 2 illustrates the average ETR for different **G** sizes across all 10 experiments. We can see that GReM is 2/4 times slower than MinorMiner with an average of 3.6 times. This means that, on average, the model requires approximately 3.6 times more time to find an embedding compared to MinorMiner. Interesting to notice is that, as the size of **G** increases, the ETR increases at a slower pace. Moreover, by analysing the experiments we noticed that for some of them the ETR even decreases for larger graphs, suggesting

that GReM may improve its performance w.r.t. to MinorMiner for bigger instances of **G**.

3.3. Ablation Studies

In this section, we outline the key components of the model and the rationale behind its design by commenting the results of some ablation studies on such components.

3.3.1 Agent

As described in section 2.4, the agent is composed of two separate GNNs, one processing the graph problem **G** and the other one the topology graph **H**. One of the early analyses showed that using only a Feed-Forward Neural Network is inefficient, so we introduced Graph Neural Networks. We analysed three different models: a simple model without Graph Neural Networks, the GCN-1 model with an agent composed by one GCN layer for **G** and one for **H** and finally the GCN-3 model composed of a single GCN layer for **G** and three GCN sequential layers for **H**. To have an idea of how good the embedding was, the results were compared with MinorMiner results. The models for the experiments were trained on fully-connected graphs, with a fixed number of nodes, mapped to a Chimera topology having fixed number of qubits. We chose to use a Chimera topology since, due to its sparsely connected structure, it is harder to find embeddings on it. We conducted experiments on multiple pairs of **G** and **H** graphs, and for each one we collected the size of the smallest embedding found during training. The experiments for **G** graphs with 8 nodes mapped on a Chimera topology with up to 128 qubits, showed the 3-GCN was the model able to find the smallest embedding close to MinorMiner, highlighting that concatenating three GCN layers helps the model understand the structure of **H** when we increase its size, and it allows to find smaller embeddings even for bigger instances of **G**.

3.3.2 Training

In the early experiments where a dataset was introduced, the dataset was composed of connected graphs with a number of nodes ranging form 3 to 7, sampled without a precise order and with an unbalanced number of graphs per size. For each **G** size the graphs were equally distributed according to the number of edges and the isomorphism classes. CL was not implemented yet and the model would simply train on the dataset until a fixed number of steps was reached. The model was tested on both Chimera and Zephyr topologies, the former with 1152 qubits, the latter with 2176 qubits. The SR obtained for both experiments was **100%**. The QUR was **50.81%** for Chimera and **34.13%** for Zephyr. This means that using this initial implementation of the dataset, MinorMiner was able to find an embedding using half the qubits of the RL model. Moreover, the performance were better on Zephyr thanks to the increased connectivity of the topology compared to Chimera.
Once we had a baseline for the dataset, we tried to improve training with the objective of minimizing the size of the embeddings found by the model on the test set. We modified the training to sample the graphs from the dataset with an increasing number of nodes, in such a way that the model would face all graphs with 3 nodes first, then all graphs with 4 nodes, and so on. Every time the model has seen the entire dataset, it would be fed again starting from graphs with 3 nodes and increasing, until all training steps are completed. The goal of this component was to help the model learn better embeddings for larger graphs, starting from simpler cases and progressively increasing the size of the problem during training. Experiments were conducted on Chimera and Zephyr topologies with the same size as in the previous case. The QUR for Chimera and Zephyr were respectively **38.27%** and **35.99%**. The results showed a significant improvement in the embeddings for Chimera, while the performance on Zephyr remained close to the performance obtained in the previous case, probably due to the higher connectivity of the topology that already helps the model in finding a good embedding for smaller graphs.
Next, we acted on the number of graphs introduced per size. We equally balanced the number of graphs sampled from the dataset for each **G** size. For smaller sizes, where the total number of graphs is very limited, the graphs are repeated until a set number is reached. The number of graphs per size and the percentage of graphs used for training is the same as the one defined for the complete model in Section

3.2. The objective is to allow the model to see enough graphs for each size to better learn how to perform the embedding on smaller graphs, by giving it more training time on them. Training was done using the same settings of the previous case. The average QUR obtained for Chimera and Zephyr are **37.48%** and **23.22%**. Results showed that balancing the number of graphs per size improves the performance of the model on the Zephyr topology, while maintaining almost constant the performance on Chimera.
Then, we wanted to evaluate the performance of the model on larger problems, so we tested it on **G** graphs with up to 11 nodes mapped on a Zephyr topology with 2176 qubits. The average QUR obtained by the model was **50.40%**, while the average SR was **99.87%**. Results showed a significant decrease of the performance on the number of qubits used for the embeddings, and an average SR slightly worse than the one obtained by the other models. To further improve the performance of the model we implemented a CL process in the training phase.
By using the CL process, we allow the model to train over graphs with a fixed number of nodes until some convergence criteria are satisfied. This is repeated for each graph size, in a increasing way, until the maximum size is reached. This method allows the model to better learn the embeddings for each size, and having the graphs ordered helps it to use the knowledge learned for smaller graphs to more easily understand how to map larger graphs. Experiment were conducted on graphs with up to 11 nodes mapped on a Zephyr topology with 2176 qubits. The model obtained a QUR of **28.72%** and a SR of **100%**. Results showed a significant improvement in the performance of the model on graphs with up to 11 nodes using Curriculum Learning. However, when trained on large graphs, the model would tend to forget how to perform embeddings on smaller graphs, decreasing its performance (QUR of **31.52%** for graphs with up to 7 nodes).
To overcome this problem, we added to the training process the re-use of smaller graphs. In other words, when training on graphs with $n > 3$ nodes, the model would randomly face graphs with $2 < m < n$ nodes, in such a way that it keeps training on small instances while learning how to embed larger graphs. For these experiments we chose to reintroduce 500 smaller graphs for each **G** size, as mentioned in Section 3.2. The obtained model is the complete implementation of this thesis and the results collected are shown in section 3.2. They show an improvement in the model to a final QUR of **20.89%**.

4. Conclusions

To conclude, we demonstrated that GReM is able to achieve results comparable to the state-of-the-art MinorMiner heuristic for problems with up to 11 nodes mapped on next generation Quantum Annealers with up to 2176 qubits. The model is able to find valid embeddings for all graphs on the test dataset, producing mappings that occupy on average around 21% more qubits than MinorMiner's embeddings across all experiments. This shows that Reinforcement Learning and Graph Neural Networks represent an innovative and promising approach to tackle this task for generic problems on Quantum Annealers. Possible future works include training the model on problem and topology graphs with larger sizes, testing other GNN architectures and adding new features to the graphs.

References

[1] Jun Cai, William G. Macready, and Aidan Roy. A practical heuristic for finding graph minors. *CoRR*, abs/1406.2741, 2014.

[2] Fred W. Glover and Gary A. Kochenberger. A tutorial on formulating QUBO models. *CoRR*, abs/1811.11538, 2018.

[3] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

[4] Tadashi Kadowaki and Hidetoshi Nishimori. Quantum annealing in the transverse ising model. *Phys. Rev. E*, 58:5355–5363, Nov 1998.

[5] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.