Neural Maximum Independent Set*

Thomas Pontoizeau, Florian Sikora, Florian Yger, Tristan Cazenave

LAMSADE, CNRS, Université Paris-Dauphine, PSL Research University
first.last@lamsade.dauphine.fr

Abstract. The emergence of deep learning brought solutions to many difficult problems and has recently motivated new studies that try to solve hard combinatorial optimization problems with machine learning approaches. We propose a framework based on Expert Iteration, an imitation learning method that we apply to solve combinatorial optimization problems on graphs, in particular the Maximum Independent Set problem. Our method relies on training GNNs to recognize how to complete a solution, given a partial solution of the problem as an input. This paper emphasizes some interesting findings such as the introduction of learned nodes features helping the neural network to give relevant solutions. Moreover, we represent the space of good solutions and discuss the ability of GNN's to solve the problem on a graph without training on it.

1 Introduction

Given a graph, finding a set of pairwise non-adjacent nodes of maximum size is probably one of the most classical hard algorithmic graph problem [20,25] denoted MAXIMUM INDEPENDENT SET. Such a problem finds natural applications to non overlapping problems, including automatic label placement for instance [2].

MAXIMUM INDEPENDENT SET can also be seen as a 1-player game where the player chooses iteratively a new node to select in order to form the largest possible independent set. Considering such an optimization problem as a game opens the possibility of applying machine learning methods such as neural networks and deep reinforcement learning. Neural networks came back to the light when it made it possible to solve difficult and large scale classification tasks in computer vision. Nowadays, the development of graph convolution layers paves the way to the application of neural networks to graph problems. If the Operational Research community studied the resolution of such graph problems for 50 years, *learning* how to solve combinatorial optimization problems has been a recent interest in the machine learning community with the emergence of scalable methods like Graph Neural Networks (GNN) [13,15,28,36].

Deep Reinforcement Learning gave state of the art results on complex two-player complete information games such as Go, Chess and Shogi [30]. In those approaches, the underlying reinforcement learning paradigm is Expert Iteration [5]. It is a general algorithm that uses Monte Carlo Tree Search to leverage the learning of a policy and a

* The authors acknowledge the support of the ANR as part of the "Investissements d'avenir" program (ANR-19-P3IA-0001, PRAIRIE 3IA Institute) and through the project DELCO (ANR-19-CE23-0016).

value. Inspired by its successes on difficult games, we borrow similar ideas to apply it to combinatorial optimization problems on graphs. In this work, we focus on learning a policy for the MAXIMUM INDEPENDENT SET problem. To do so, we use Monte Carlo Search to generate diversified and well-explored states of this problem for each instance and use a Graph Neural Network to learn the resulting policy.

In this paper, we investigate several ingredients giving the ability to GNN to learn how to solve combinatorial optimization problems. Our main contributions are threefold, first, we show that learning additional node features boosts the performance of GNNs. Secondly, we show how such a neural network can be used to generate a diversified pool of good solutions, giving useful informations about the distribution of the best solutions and potentially indicating how hard it is to find the optimal solution. Finally, we show how to transfer knowledge between graphs, training a GNN on a small graph and applying it successfully on a larger graph. Although we primarly focus on MAXIMUM INDEPENDENT SET problem, it is merely a case study and our framework is versatile enough to be adapted to other cases.

First, we introduce the relevant literature about solving combinatorial optimization problems with neural networks in Section 2. Then, in Section 3, we present the basic concepts and notations used throughout this paper. In Section 4, we provide the details about our method and how the results were obtained. The results of our numerical experiments are discussed in Section 5. Finally, we end the paper with concluding remarks and a roadmap for future works.

2 Related Work

Combinatorial optimization problems in graphs like MAXIMUM INDEPENDENT SET are widely studied in many fields and can be viewed from many different perspectives.

Our paper focuses on MAXIMUM INDEPENDENT SET which is a hard problem to solve. In terms of theoretical complexity, the problem is known to be NP-hard [20], *i.e.* it is not believed that there exists any polynomial algorithm for efficiently finding its optimal solution. Even from the approximation point of view, the problem is hard to approximate [25]. However it is possible to solve MAXIMUM INDEPENDENT SET by exact exponential approaches like divide-and-conquer methods [32] or branch-and-bounds methods [8,26]. Some efficient heuristics, although coming without any guarantee on the quality of the solution, have been devised [14,22].

Recently, several approaches tried to solve combinatorial optimization problems by using machine learning techniques [7,21,24]. It gained momentum with the emergence of Graph Neural Networks and their expressive power [13,36] and it motivated researchers to investigate how to apply GNN to graph combinatorial optimization problems [18,15].

For example, Abe et al. applied Alpha Zero for several combinatorial optimization problems, including MAXIMUM INDEPENDENT SET [1]. Other works combined branch-and-bound methods with GNNs [16]. In [19], the authors introduced an unsupervised learning framework inspired by Erdős probabilistic method. Reinforcement learning has also been investigated [12]. In [21], the authors proposed a framework

combining reinforcement learning and graph embeddings in order to incrementally construct the solution.

In our work, we build a framework that solves combinatorial optimization problem based on Expert Iteration [5], a promising reinforcement algorithm based on imitation learning. Our approach is innovative as it introduces learned features that help to improve the performance of the GNNs.

On the other hand, it has to be noted that GNNs are not always working perfectly. More precisely, it has been shown that stacking too many layers was detrimental as it lead to over-smoothing, *i.e.* the features of the nodes are propagating too far in the network and all information on each node vanishes [10]. In addition, when solving a combinatorial optimization problem in a supervised manner, some difficulties have been highlighted for generating unbiased representative labels for large scale instances [35]. In that way, we investigate transferring knowledge from small instances to larger instances and give encouraging results for future work.

3 Preliminaries

The Maximum Independent Set combinatorial optimization problem. In the remainder of this paper, we denote by $G = (V, E)$ an undirected graph in which V (resp. E) is the set of nodes (resp. edges). The distance between two nodes is the number of edges in the shortest path connecting them.

Given a graph $G = (V, E)$, an independent set is a set $S \subset V$ of nodes such that $\forall i, j \in S, i \neq j \Rightarrow \{i, j\} \notin E$. The problem MAX INDEPENDENT SET consists in finding an independent set of maximum size.

Seeing Max Independent Set as a game. It is possible to interpret MAX INDEPENDENT SET as a game in which the goal is to find a set of vertices of maximum cardinality by iteratively selecting a node under the constraints that the current set of selected nodes always forms an independent set. The game ends when there is no more node that can be selected and the score is the number of selected nodes.

In order to make the reading clear, we introduce some notations. All following definitions are always considering a given undirected simple (without any multiple edges nor any self-loops) graph $G = (V, E)$.

A *state* is a partial solution for MAX INDEPENDENT SET in G, *i.e.* a set of nodes $S \subset V$ that forms an independent set, not necessarily of maximal size. We call *initial state* the state in which no node has been selected, and *final state* a state of maximal size, *i.e.* a state in which no more nodes can be added to the solution without compromising the constraint of being an independent set. Sometimes, final states are naturally called *solutions*.

Given a state, a *legal move* is a node that does not belong to the state and can be added to it such that the whole set still forms an independent set.

Given a state, a *sequence* is a set of nodes not belonging to the state that can be added to it and form a final state. We call *labeled state* a couple $\{$state, sequence$\}$.

Given a state, a *policy* for the legal moves is a distribution of their probability to be selected.

Given a state and a method to obtain a policy for any state, a *rollout* is a sequence obtained by choosing nodes iteratively according to probability provided by the method.

Graph Neural Networks. A Graph Neural Network (GNN) is a neural network that contains graph convolutional layers that leverages the structure of a graph [36]. A graph convolutional layer takes the features of a node as an input and usually aggregates transformed features from the node and its neighbors into a set of output features. One of the major advantages of GNNs compared to other standard neural networks is that a GNN layer can be applied to any graph of any size.

In our neural networks, we used convolutional layers from *Graph Isomorphic Network* (GIN) [34] that have been shown to have good empirical performances. Note that we also tested Graph Attention Networks (GAT) [31] layers with a substantial drop in performance.

Expert Iteration. In imitation learning, an *expert* (usually represented by a tree search) creates a dataset of states with target moves drawn by him. Then, a *learner* (usually represented by a neural network) tries to imitate the behavior of the expert by predicting the policy among the moves. Expert Iteration [5] is a reinforcement learning algorithm based on imitation learning that repeats the process and improves the tree search at each iteration by using the learner policy to guide search during the dataset creation and the tree search, increasing the performance of the expert.

4 Method

4.1 Using Expert Iteration to solve Max Independent Set

Labeling phase. The first step of our algorithm consists in produces a set of labeled solutions that we attempt to predict in the learning phase.

First, the expert generates 500 random states that are not final with a uniform policy (*i.e.* probability distribution among the current legal moves). Then, for each randomly generated state, we designed the expert to make 50 random rollouts and keep the one with the best score as the sequence in the labeled state. Those settings allow the expert to provide a training set in short time and still give meaningful results for our observations, and turns out to be enough to reach the state-of-art in terms of quality of the solution for the studied datasets.

This naïve Monte Carlo search approach has the convenience to be very quick to execute and still to allow to highlight the quality improvement of the learning.

Learning phase. The set of labeled states constitutes the training set of the GNN. Our learner is a graph neural network that we train on the training set, allowing to compute outputs for any state. Given a state, the learner tries to predict the nodes to select to complete the partial solution by mimicking the expert plays. Therefore, each input of the GNN represents a state of the game. To enrich the input, each node can be described by the following four possible groups of features :

- a $0-1$ feature *state* indicating if the node belongs to the current state or not.
- a $0-1$ feature *legal_moves* indicating if the node is a legal future move considering the current state.

- two random features *footprints* belonging to $[0, 1]$ that characterizes uniquely each node. This relates to the use of random features that have been proved to be helpful for GNN learning for solving other combinatorial problems like MINIMUM DOMINATING SET [29] by singularizing the nodes and breaks symmetries.
- two features *distances* indicating the number of nodes at distance 1 (resp. 2) from it.

Each group of feature is normalized according to the ℓ_2-norm.

In our architecture, we added the possibility to concatenate two additional features to the input for each nodes before the first layer that are learned during the training (*i.e.* those features are considered as weights of the neural network). We call them *node embeddings*. Thus, when we put all features for each node, each input tensor has a shape $(|V|, 8)$.

Our neural network has the following architecture. One block consists of one Conv1D layer (linear transformation that expands each node features separately), one GIN layer (containing ReLu layers and batchnorm layers), one dropout layer, and one Conv1D layer. The network consists of k blocks, adding the initial residual (copy of the initial input) to the result between each block, and finally apply a final sigmoid. In our experiments, GAT layers instead of GIN ones did not really give better performance and two blocks were usually enough. Adding the initial residual has been motivated in [11]. The target is a one-hot-vector of the nodes of the sequence given by the labeling phase. We then naturally evaluate the loss between the target and the output using the Binary Cross-Entropy loss.

Note that at the beginning of each learning phase, we reset the parameters of the neural network. It makes sense since the expert plays are better each iteration. Once the GNN is trained, it can be used as a policy generator by applying a softmax function to the output for any given state, defined as: $softmax(z, t)_i := e^{\frac{z_i}{t}} / \sum_{k\in\{1\dots n\}} e^{\frac{z_k}{t}}$, with $z = (z_k)_{k\in\{1\dots n\}}$ the output of the model, t the temperature (which is a parameter). Notice that the greater t is, the closest to uniform policy the softmax function is. When t is close to 0, the softmax function tends to give the maximum value of the output.

Our approach consists in repeating the labeling phase and the learning phase, using the learner to perform rollouts instead of using a uniform policy for the labeling phase. Except for `dimacs-frb30-15-1`, our results were obtained within actually only one iteration (one labeling phase and one learning phase).

4.2 Evaluation of the neural network

In order to control how relevant is our neural network, we introduce two methods to evaluate it.

- **Argmax sequence method.** Starting with the initial state, compute the output of the neural network on the current state and choose the node with the highest activation among the legal moves. Iterate until the state is final. We call such a sequence the *argmax sequence*.
- **Stochastic exploration method.** To produce one solution, start with the initial state (*i.e.* the state in which no node has been chosen yet in the solution.), compute

the output of the neural network on the current state and apply the softmax function with a low temperature that we call the *exploration temperature* (in our experiments, we used a temperature of 0.106). Then, the next node is chosen according to the distribution of probabilities given by the softmax function among legal moves. Iterate until the state is final. Repeat the previous procedure k times to obtain a pool of k solutions as the result of the evaluation.

The latter gives richer information about the neural network performance, but is more than k times slower then the former. We sometimes use the first method as a simple and quick way to observe the quality of the neural network at each moment of the learning phase.

We tuned the hyper-parameters with Optuna [3]. The optimized algorithm to do so consisted in one labeling phase, one learning phase and one stochastic exploration for evaluation. The simplest objective function we could use is the score of the best found solution during the stochastic exploration, but we wanted to refine it by adding a notion of diversity in the pool of best solutions. In order to do so, we optimized the quality q of the set of the solution with the best found score, according to the following definition: $q = (b + 1) - 1/(\sqrt{|S|} \cdot trace(covariance(S^T)))$, with b the best score found during the stochastic exploration and S the set of found solutions with score b. In this way, the best found score is always highlighted by the objective function, but some bonus is given if the set of solutions with such score is large and diverse.

Our hyper-parameters suggested by Optuna comprise a very low exploration temperature (0.106), a dropout value of 0.2, and 145 epochs.

5 Experimental Results

In this section, we present interesting observations about learning how to solve MAX INDEPENDENT SET with GNNs. We highlight our experimental results among the following instances of graphs :

instance	$\|V\|$	$\|E\|$	best known score
`ba200_5`	200	975	82
`er200_10`	200	1957	41
`dimacs-frb30-15-1`	450	17827	30
`bio-SC-LC`	2004	20452	968

`er200_10` is a standard generated Erdős-Renyi graph that have edge probability $p = 0.10$ and `ba200_5` is a Barabási-Albert generated graph in which 5 edges has been added per node [6].
`dimacs-frb30-15-1`[1] is a difficult instance from the Benchmarks with Hidden Optimum Solutions for Graph Problems [33] provided in [1] `bio-SC-LC` is a real-world instance from [27]. These four instances are a subset of instances used in [1].

[1] we kept the same name used in [1] but it is actually an instance of BHOSLIB [33]

5.1 Boosting the learning phase with good features.

In this subsection we emphasize two major points. First, GNNs can be trained on a set of quickly labeled states and perform well to solve MAX INDEPENDENT SET. Second, adding embeddings to node features boosts the learning phase and helps the GNN to learn more efficiently.

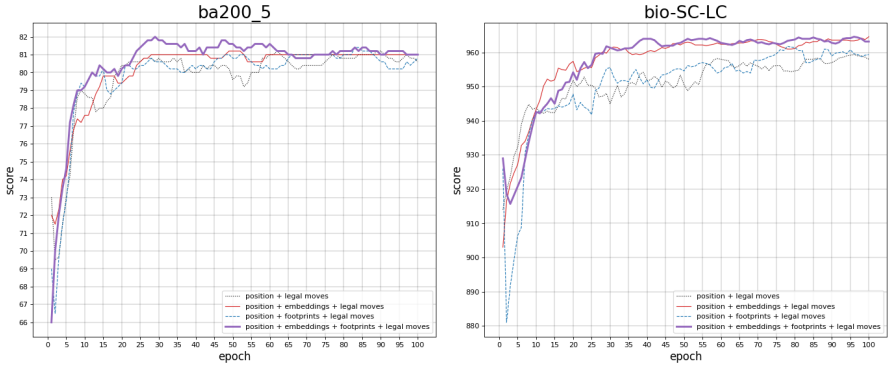

Fig. 1: The score of the argmax playout each epoch of the learning on each instance. Learning curves with embeddings are highlighted by a plain curve.

In Figure 1, our GNN learns the label of a set of 500 labeled states given by the labeling phase for 100 epochs on each instance of our dataset (results for other instances can be found in appendix). At each epoch, we evaluate the GNN by calculating the score of the argmax sequence. We represent some curves with different ways to select footprints and embeddings among the features we introduced in last section. In order to make the graphic more readable, the curves have been smoothed by representing the average score of the 5 last scores for each epoch.

Since adding randomized features to the nodes has been observed quite efficient to singularize the nodes and to break symmetries [29], we included those new features as *footprints*, and also suggested to use the same idea allowing such footprint to be learned by the neural network (calling them *embeddings*). It appears clearly that all curves describing the performance of a network involving embeddings give better performance than ones without, and perform more efficiently than standard randomized features (see Fig. 1). On `dimacs-frb30-15-1`, our GNN showed more difficulty to learn but eventually succeeded on this difficult instance.

Those observations are based on a quick evaluation but can be made more precisely by using the second way to evaluate the GNN. In Figure 2, we represented all the scores found in the exploring phase (in gray), the ones found by making playouts from the initial state with a GNN that trained on inputs with the state as a unique feature in light blue (resp. all features in dark blue). Resultas for other instances can be found in the appendix. The three sets contain 500 solutions each. Our observation show that the GNN does not succeed in beating the scores of the expert on `dimacs-frb30-15-1`.

We give some interpretation in next subsections based on the cluster of best found solutions. Overall, using all features clearly appears to be a good setting and outperform the expert plays and the stochastic exploration with no extra features.

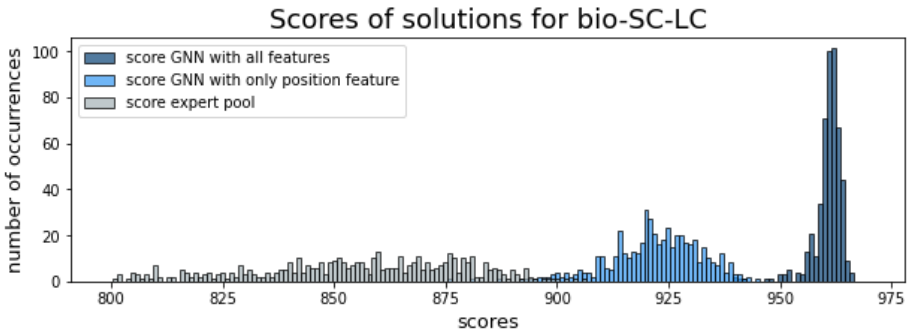

Fig. 2: Distribution of scores of 500 solutions from the training set and an exploration of 500 rollouts with the GNN for instance `bio-SC-LC`, first with only the state, and then with all the features.

Once a neural network has learned on a small graph, it performs a little better on other bigger instances it never encountered, removing the footprint features and the embedding features during the learning phase since they had sense only in a particular instance. However, the quality of the learning is quite less effective than with all features. A discussion about this observation can be found in the appendix.

6 Conclusion

In this paper, we combined GNN with expert iterations to learn how to solve MAX INDEPENDENT SET problems and improved the overall network architecture with learned node features. We brought empirical evidences of the soundness of our approach and gave some qualitative results about the space of solutions explored by our approach.

On standard instances, GNNs seem to learn correctly how to solve MAX INDEPENDENT SET and are able to give good solutions with some easy generated training set. Our method seems to performs well on real-world instances but shows some difficulties on artificially created hard instances. Besides, we emphasized some interesting observations about the improvement made by adding embedding features to nodes. This could be investigated further in future work. In particular, we do not know yet what those embeddings are exactly learning, how meaningful they are, and how much they can be exported for other tasks.

Furthermore, our framework is general and could fit many problems such as MIN VERTEX COVER, MAX CLIQUE, or the weighted versions of them (by simply putting the weight instead of a $0-1$ activation in the state feature).

References

1. Abe, K., Xu, Z., Sato, I., Sugiyama, M.: Solving NP-Hard Problems on Graphs with Extended AlphaGo Zero (2020)
2. Agarwal, P.K., van Kreveld, M., Suri, S.: Label placement by maximum independent set in rectangles. Computational Geometry **11**(3), 209–218 (1998)
3. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: A Next-Generation Hyperparameter Optimization Framework. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. p. 2623–2631. KDD '19, Association for Computing Machinery, New York, NY, USA (2019)
4. Alexander, J., Mink, T.: A New Method for Enumerating Independent Sets of a Fixed Size in General Graphs. Journal of Graph Theory **81**(1), 57–72 (2016)
5. Anthony, T., Tian, Z., Barber, D.: Thinking fast and slow with deep learning and tree search. In: Advances in Neural Information Processing Systems. pp. 5360–5370 (2017)
6. Barabási, A.L., Albert, R.: Emergence of Scaling in Random Networks. Science **286**(5439), 509–512 (1999)
7. Bengio, Y., Lodi, A., Prouvost, A.: Machine learning for combinatorial optimization: A methodological tour d'horizon. European Journal of Operational Research **290**(2), 405–421 (2021)
8. Bourjolly, J.M., Laporte, G., Mercure, H.: A combinatorial column generation algorithm for the maximum stable set problem. Operations Research Letters **20**(1), 21–29 (1997)
9. Byskov, J.M.: Enumerating maximal independent sets with applications to graph colouring. Operations Research Letters **32**(6), 547–556 (2004)
10. Chen, D., Lin, Y., Li, W., Li, P., Zhou, J., Sun, X.: Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020. pp. 3438–3445. AAAI Press (2020)
11. Chen, M., Wei, Z., Huang, Z., Ding, B., Li, Y.: Simple and deep graph convolutional networks. In: III, H.D., Singh, A. (eds.) Proceedings of the 37th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 119, pp. 1725–1735. PMLR (13–18 Jul 2020)
12. Chen, X., Tian, Y.: Learning to perform local rewriting for combinatorial optimization. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 32. Curran Associates, Inc. (2019)
13. Chen, Z., Chen, F., Zhang, L., Ji, T., Fu, K., Zhao, L., Chen, F., Lu, C.T.: Bridging the gap between spatial and spectral domains: A survey on graph neural networks (2020)
14. Das, K.N., Chaudhuri, B.: Heuristics to find maximum independent set: An overview. In: Deep, K., Nagar, A., Pant, M., Bansal, J.C. (eds.) Proceedings of the International Conference on Soft Computing for Problem Solving (SocProS 2011) December 20-22, 2011. pp. 881–892. Springer India (2012)
15. Dwivedi, V.P., Joshi, C.K., Laurent, T., Bengio, Y., Bresson, X.: Benchmarking Graph Neural Networks. arXiv e-prints (2020)
16. Gasse, M., Chételat, D., Ferroni, N., Charlin, L., Lodi, A.: Exact combinatorial optimization with graph convolutional neural networks. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada. pp. 15554–15566 (2019)

17. Gurski, F., Rehs, C.: Counting and enumerating independent sets with applications to combinatorial optimization problems. Mathematical Methods of Operations Research **91**(3), 439–463 (2020)
18. Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., Leskovec, J.: Open graph benchmark: Datasets for machine learning on graphs (2021)
19. Karalias, N., Loukas, A.: Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds.) Advances in Neural Information Processing Systems. vol. 33, pp. 6659–6672. Curran Associates, Inc. (2020)
20. Karp, R.M.: Reducibility among Combinatorial Problems, pp. 85–103. Springer US (1972)
21. Khalil, E., Dai, H., Zhang, Y., Dilkina, B., Song, L.: Learning combinatorial optimization algorithms over graphs. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc. (2017)
22. Lamm, S., Sanders, P., Schulz, C., Strash, D., Werneck, R.F.: Finding Near-Optimal Independent Sets at Scale. Journal of Heuristics **23**(4), 207–229 (2017)
23. Van der Maaten, L., Hinton, G.: Visualizing data using t-sne. Journal of machine learning research **9**(11) (2008)
24. Mazyavkina, N., Sviridov, S., Ivanov, S., Burnaev, E.: Reinforcement learning for combinatorial optimization: A survey (2020)
25. Piotr, B., Marek, K.: On some tighter inapproximability results. Tech. rep. (1999)
26. Rossi, F., Smriglio, S.: A branch-and-cut algorithm for the maximum cardinality stable set problem. Operations Research Letters **28**(2), 63–74 (2001)
27. Rossi, R.A., Ahmed, N.K.: The Network Data Repository with Interactive Graph Analytics and Visualization. In: Bonet, B., Koenig, S. (eds.) Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA. pp. 4292–4293. AAAI Press (2015)
28. Sato, R., Yamada, M., Kashima, H.: Approximation Ratios of Graph Neural Networks for Combinatorial Problems. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 32. Curran Associates, Inc. (2019)
29. Sato, R., Yamada, M., Kashima, H.: Random Features Strengthen Graph Neural Networks. CoRR **abs/2002.03155** (2020), https://arxiv.org/abs/2002.03155
30. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., Hassabis, D.: A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science **362**(6419), 1140–1144 (2018)
31. Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph Attention Networks. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings (2018)
32. Xiao, M., Nagamochi, H.: Exact algorithms for maximum independent set. Inf. Comput. **255**, 126–146 (2017)
33. Xu, K.: BHOSLIB: Benchmarks with Hidden Optimum Solutions for Graph Problems (Maximum Clique, Maximum Independent Set, Minimum Vertex Cover and Vertex Coloring) . http://sites.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm
34. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How Powerful are Graph Neural Networks? In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019 (2019)
35. Yehuda, G., Gabel, M., Schuster, A.: It's Not What Machines Can Learn, It's What We Cannot Teach. In: III, H.D., Singh, A. (eds.) Proceedings of the 37th International Conference on

Machine Learning. Proceedings of Machine Learning Research, vol. 119, pp. 10831–10841 (2020)
36. Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., Sun, M.: Graph neural networks: A review of methods and applications (2019)

A Results on other instances

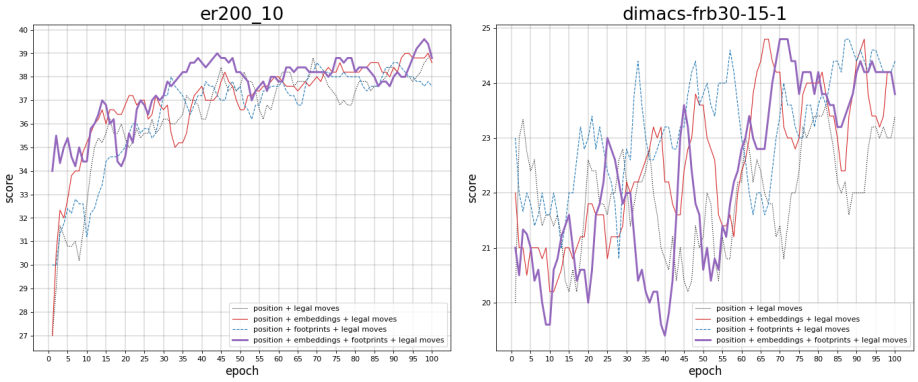

Fig. 3: The score of the argmax playout each epoch of the learning on each instance. Learning curves with embeddings are highlighted by a plain curve.

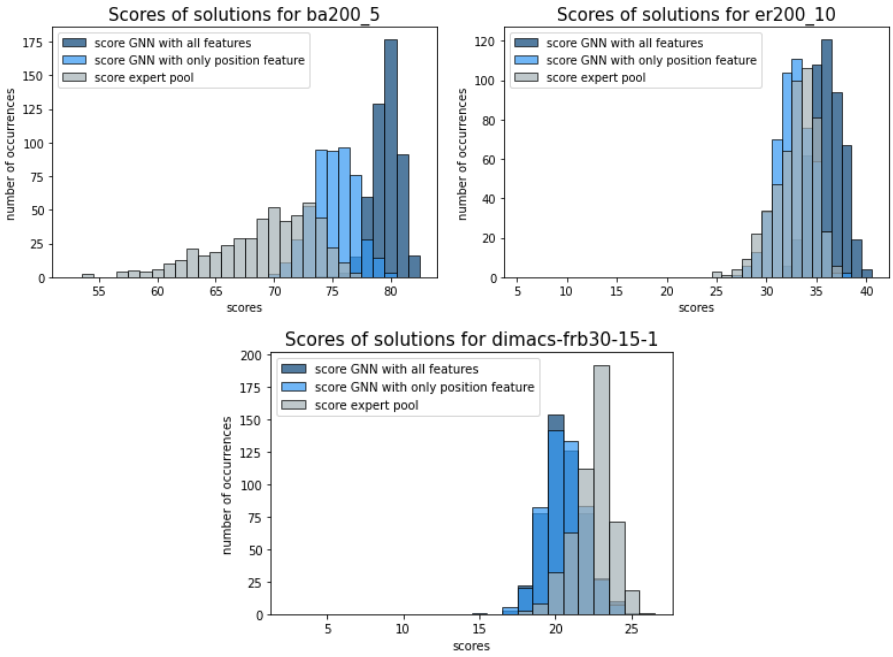

Fig. 4: Distribution of scores of 500 solutions from the training set and an exploration of 500 rollouts with the GNN for each instance, first with only the state, and then with all the features.

B Stochastic exploration method allows to find various good solutions

By giving a set of good solutions, the stochastic exploration method gives us some interesting insights about the distribution of the solutions in a given instance. After several exploration methods, we saved all good solutions of the three best found scores in order to observe the clusters they form (*i.e.* in the sense of hamming distance between sets). Note that the pool of best solutions for `dimacs-frb30-15-1` is the only one that necessitated 5 iterations of labeling/learning phase and the solutions were found in the labeling phase, the stochastic exploration method giving poor solutions as discussed previously.

We sum up the information about the set of solutions we obtained:

- `ba200_5`: 1400 solutions with score 80, 700 solutions with score 81, 123 solutions with score 82.
- `er200_10`: 1400 solutions with score 39, 700 solutions with score 40, 94 solutions with score 41.
- `dimacs-frb30-15-1`: 499 solutions with score 26, 77 solutions with score 27, 4 solutions with score 28.
- `bio-SC-LC`: 21 solutions with score 966, 5 solutions with score 967, 1 solution with score 968.

In order to observe how the solutions are organized, we first computed all hamming distances between each pair of solution to see how far they are from each other. In Figure 5, we represented the distribution of all possible hamming distances for the three best found scores for each instance. For `er200_10` and `dimacs-frb30-15-1`, we also represented the graphic in log scale for more readability.

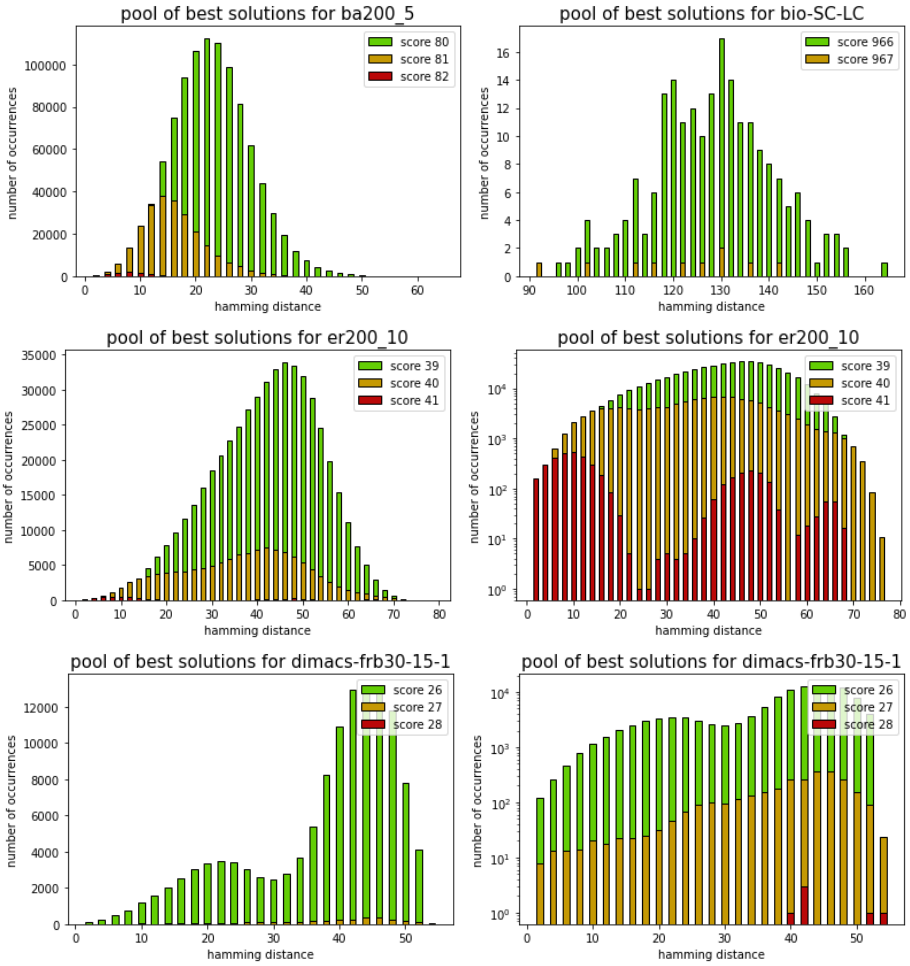

Fig. 5: The distribution of hamming distances between best found solutions for each instance of the dataset. Sometimes we also give the same graphic in log scale for more readability.

For `ba200_5`, the solutions are well organized along a Gaussian curve. For `er200_10`, we observe almost the same phenomenon except that the best solutions (with score 41) are split into three clusters : one big cluster of close solutions, one sparse cluster of partially close solutions, and a small sparse cluster of solutions. For `dimacs-frb30-15-1`, all best found solutions look very sparse and have very few nodes in common. For `bio-SC-LC`, the solutions seem to be well organized and not so far from each other. Note that there is no hamming distance for the score 968 since we only obtained one solution for this score.

For each instance, we embedded our pool of solutions into a three dimensional space with a t-SNE [23] in Figure 6, that allows to observe our previous remarks on the distribution of hamming distances. We can notice that the scatterplots are always nested around the best solutions.

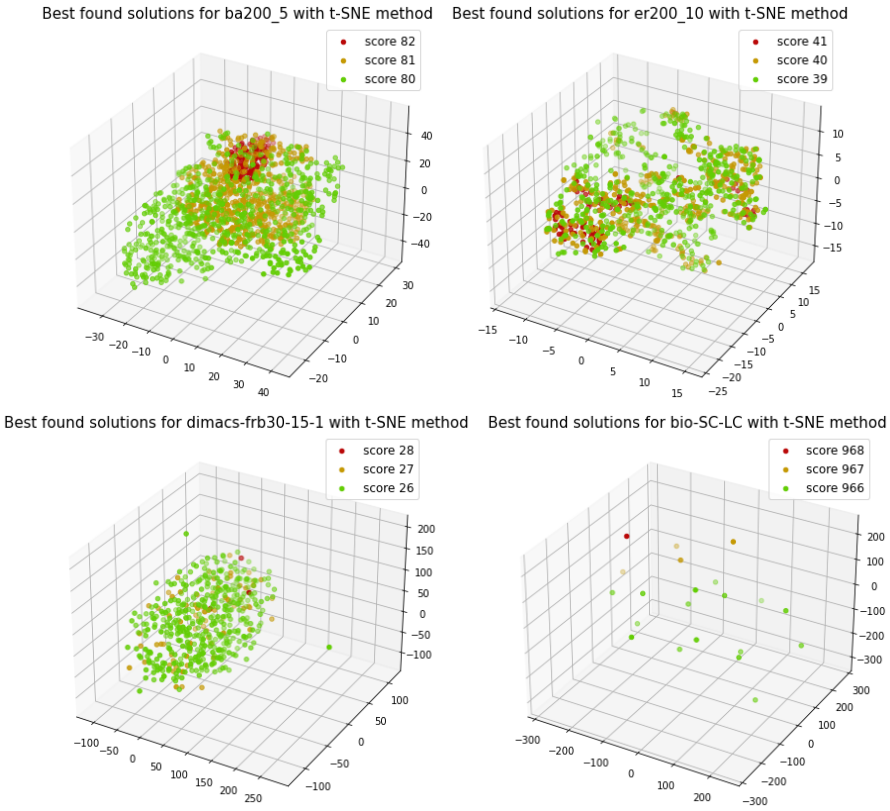

Fig. 6: The projection of the set of best found solutions for each instance with the t-SNE method.

Note that for the instances `ba200_5` and `er200_10`, we had to sample the second and third best found solutions (in green and orange) in order to highlight the pool of best solutions (red).

With the instance `dimacs-frb30-15-1`, we observe that all best found solutions look very sparse in the sense that the solutions have very few nodes in common. This could explain why GNN cannot converge properly into a good solution. The latter remark could explain why our neural network has so much difficulty to learn how to solve the problem on this instance and converge to good solutions.

Since enumerating maximal or maximum independent sets has also been studied in the literature [4,9,17], our method provides some interesting tools to obtain good various solutions with the help of a GNN.

C GNNs can export expertise on Max Independent Set from a small graph to a larger graph

The last observation we want to highlight is that once a neural network has learned on a small graph, it performs a little better on other bigger instances it never saw.

In order to do so, we had to remove the footprint features and the embedding features during the learning phase since they had sense only in a particular instance. Thus, the quality of the learning is quite less effective than with all features.

In our experiment, we compare training on `ba200_5` and `er200_10` and see how much the GNN performs on the other instances (including two other instances : `ba100_5` and `ba1000_5` constructed in the same way than `ba200_5`).

After a labeling phase, we make a learning phase and compute the score of the argmax sequence for each other instance using the trained model at each epoch. We then smoothed the curves by computing the average score of the last 5 scores at each epoch, and then represented the approximation ratio between the score of the argmax sequence and the best known result on the instance for each epoch. The resulted curves are represented in Figure 7.

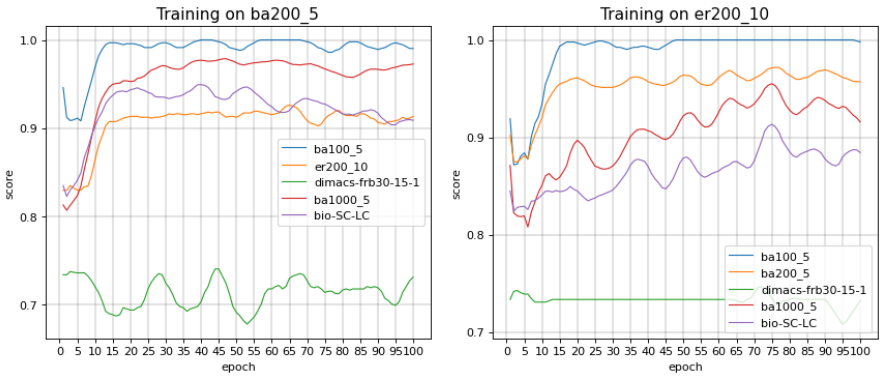

Fig. 7: Evolution of approximation ratio (compared to the best known score) on the argmax playout on other graph training on ba200_5 and er200_10.

When training on `ba200_5`, the quality of the argmax sequence improves quickly at the beginning and then stagnates. When training on `er200_10`, the qualitywork of the argmax sequence increases slower than with `ba200_5` but increases all along the learning. On the other hand, not surprisingly, note that the learning phase did not make any improvement on `dimacs-frb30-15-1`. Those observations indicate that

our GNN was able to transfer some knowledge about MAXIMUM INDEPENDENT SET on brand new graphs, and is promising for future work.