OPT2024: 16th Annual Workshop on Optimization for Machine Learning

Graph Neural Networks for Hyperparameter Inference in Ising Solvers

Edward Jiang EDWARD.JIANG@NTT-RESEARCH.COM

Samuel Reifenstein SAMUEL.REIFENSTEIN@NTT-RESEARCH.COM
Milin Doppalapudi MILIN.DOPPALAPUDI@NTT-RESEARCH.COM
Timothee Leleu TIMOTHEE.LELEU@NTT-RESEARCH.COM
NTT Research Inc.

Abstract

We propose a novel method to apply graph neural networks (GNNs) to combinatorial optimization problems. Unlike existing approaches that use GNNs to directly solve problem instances, our method instead uses them to predict hyperparameters for a heuristic solver. The model is trained in a supervised fashion on a small dataset of graphs, with corresponding hyperparameters obtained through conventional hyperparameter optimization routines. During inference, the model predicts near-optimal hyperparameters for unseen instances that minimize the runtime of the heuristic solver. Experiments show that our method generalizes well to much larger graphs, and outperforms manually hand-tuned parameters. The framework is flexible and can be applied to a wide variety of combinatorial optimization problems or heuristic solvers.

1. Introduction

Due to their discrete and highly non-convex nature, problems in combinatorial optimization (CO) are generally intractable from a complexity theory perspective to solve exactly [15]. Because of this, countless heuristic algorithms have been proposed that find near-optimal or approximate solutions, such as branch-and-bound algorithms [26, 27, 34], Markov chain Monte Carlo methods (simulated annealing [17], parallel tempering [37]), continuous-state dynamical systems (simulated bifurcation machines [9], chaotic amplitude control [19, 31, 32]), or modern boolean satisfiability (SAT) solvers [4, 24, 35],

A series of recent works have focused on applying methods from machine learning to solve combinatorial optimization problems, with several of them relying on graph neural networks (GNNs) as a backbone[6, 29, 36, 41]. However, it has been argued that the vast majority of these models, if not all of them, are outperformed by a simpler heuristic algorithm such as simulated annealing or even greedy search [2, 3]. In addition, there have been many works recently which use GNNs to do algorithm selection for combinatorial optimization [11, 45] with some success at improving solving times. Our method differs in the sense that we focus on a specific heuristic algorithm and try to find good hyperparameters for this algorithm.

In this paper, we propose a novel method to use GNNs for combinatorial optimization. Instead of using a GNN to solve the problem itself or select an algorithm from a list of solvers, we use it to predict hyperparameters for another specific heuristic solver. In the offline phase, the GNN is trained using supervised learning on a dataset composed of pairs of input graphs generated from various random distributions, along with optimal hyperparameters obtained via an automated tuning

algorithm. When a new, unseen problem is presented (or online phase), the GNN rapidly infers near-optimal parameters tailored to the problem type, significantly reducing the running time of the heuristic algorithm. The time required for parameter inference is significantly less than the runtime of the CO solver, allowing us to exploit prior knowledge about the correlation between problem structure and good parameters. We show that our method is able to generalize well to graphs much larger than those seen during training.

2. Automatic Hyperparameter Selection

2.1. Problem Statement

Given an instance G to a CO problem and a heuristic solver with hyperparameters $\boldsymbol{\alpha}$ and an associated computational cost function $\mathcal{L}(G; \boldsymbol{\alpha})$, the set of optimal hyperparameters for G is given by

$$\boldsymbol{\alpha}_G^* = \arg\min_{\boldsymbol{\alpha}} \mathcal{L}(G; \boldsymbol{\alpha}) \tag{1}$$

The problem of automatic hyperparameter selection is to learn the mapping $G \mapsto \boldsymbol{\alpha}_G^*$.

For small instances G or instances with a particular structure, equation (1) can be solved using standard hyperparameter optimization routines. However, learning the mapping for general instances G is generally intractable and remains yet to be solved.

2.2. Proposed framework

Our method for inference of good parameters for the solver can be divided into two submodules:

1. A hyperparameter optimization module is performed per instance of the problem which determines a set of parameters that minimizes the computational cost of the solver. We measure computational cost in terms of the expected time-to-solution (TTS) to find the optimal solution with 99% probability, which given by

   $$\text{TTS} = K \cdot \frac{\log(1 - 0.99)}{\log(1 - p_0)} \tag{2}$$

   where p_0 is the probability that the solver finds the optimal solution in a single trial, and K is some measure of the cost of a single trial, for example the number of matrix multiplications. We perform this optimization for all small instances where the optimal solution is known (for example, by using brute-force search).

   This results in a dataset containing instances of the Ising problem and a set of good parameters for the solver for each instance.

2. A deep neural network that is trained on the data generated above. Since our instances are graphs, we use a GNN to map an input graph to a set of parameters for the solver. Here we face the additional challenge of allowing the neural network to generalize to larger graphs, as we are primarily interested in these graphs where the optimal solution is not known.

   To resolve this issue, we use a learnable scaling function for some hyperparameters. This is discussed in more detail in section 2.4.

A schematic of our method is shown in Figure 1.

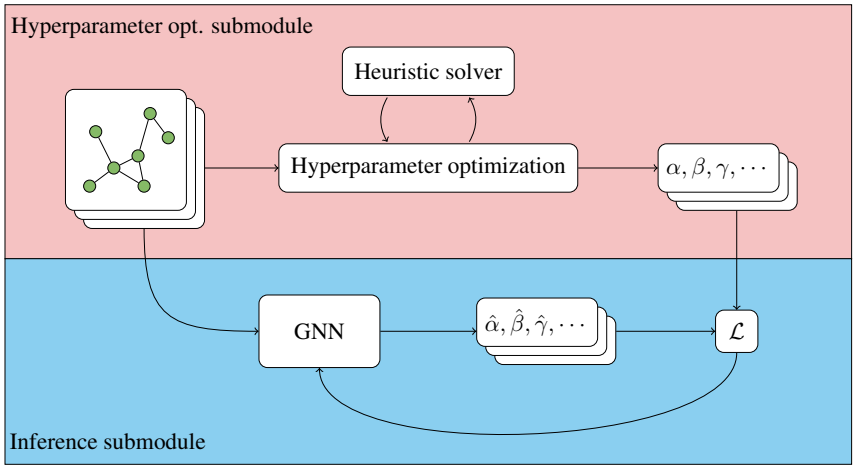

Figure 1: Our proposed framework for automatic hyperparameter selection. Greek letters here represent the hyperparameters of the solver, and $\mathcal{L}$ is the loss function between the ground-truth and the GNN predictions.

2.3. Hyperparameter Optimization Submodule

A large number of algorithms for hyperparameter optimization appear in the literature. Among the simplest include grid search [18] or random search [1], but more sophisticated ones include model-based algorithms such as Bayesian optimization with random forests [13] or tree-structured parzen estimators [42], evolutionary or population-based algorithms [21, 40], or budget-based algorithms [8, 22].

In this paper we choose to use the BOHB algorithm [8] because of its ability to handle both discrete and continuous hyperparameters, its budget-based resource allocation method, and its provable bounds on optimality. An open-source implementation is freely available via the `hpbandster`[1] package for Python.

We run the entire BOHB algorithm separately on each graph in our dataset. In each iteration, the algorithm proposes a new set of hyperparameters and allocates a resource budget, runs the solver using those hyperparameters, and records the TTS (see equation (2)). The final set of hyperparameters which we use for our ground-truth dataset is the one that i) is allocated the maximum allowed budget and ii) has the lowest TTS among all such runs.

2.4. Inference Submodule

With a ground-truth dataset generated using hyperparameter optimization, we can now do supervised learning to predict hyperparameters for previously unseen instances. To this end, we use a graph neural network (GNN), a specialized class of neural architecture that deal specifically with graph-structured data. Though many architectures exist within this class [16, 30, 39, 43], we opt to use one of the simplest, a graph convolutional network [16].

More specifically, our model architecture consists of the following:

1. https://github.com/automl/HpBandSter

i. A number of graph convolutional layers, where the output node features $X^{(\ell)}$ are given in terms of the input node features $X^{(\ell-1)}$ as

$$X^{(\ell)} = \sigma\left(\tilde{A}X^{(\ell-1)}W^{(\ell)}\right) \tag{3}$$

where $W^{(\ell)}$ is a weight matrix, $\tilde{A} = D^{-1/2}(A+I)D^{-1/2}$ is the normalized adjacency matrix of the graph, D is the degree matrix, and σ is any non-linear activation function. The initial features $X^{(0)}$ are chosen randomly and fixed.

ii. A readout layer that averages over the final node features before applying a dense layer:

$$Z = \sigma\left(W \cdot \text{AvgPool}(X^{(L)}) + b\right) \tag{4}$$

iii. A post-processor that scales the readout as a trainable function of the graph size:

$$\hat{Z} = Z \odot f(N; a) \tag{5}$$

where $f : \mathbb{N} \to \mathbb{R}^{|Z|}$ is a trainable function parametrized by a, N is the number of nodes in the graph, and $\odot$ denotes the element-wise product.

Some hyperparameters, e.g. the number of solver iterations, may have some scaling behaviour with respect to the size of the graph. Since the ground-truth dataset contains comparatively smaller instances than real-world graphs, the parameters in the dataset may not be representative of those. The post-processing step accounts of this by learning how the parameters scale as a function of the size of the graph, and so is more able to generalize by extrapolating the scaling beyond the sizes seen in training. The scaling behaviour itself can be learned by varying the sizes of the graphs in the dataset.

The function $f(N; a)$ in equation (5) can be parametrized in any way. In this paper we choose to either use the identity function or an exponential function $a_0 N^{a_1}$, depending on the hyperparameter. More sophisticated methods include parameterizing a Taylor expansion in N, or using neural arithmetic logic units (NALUs) [38] or its variants [12, 23]. These parameters are trained in conjunction with the rest of the model parameters.

The model is trained via gradient descent to minimize the mean-squared error between the predicted hyperparameters and the ground-truth hyperparameters found by the first submodule.

3. Experiments

3.1. Setting

We evaluate our framework on a heuristic solver for the Ising problem known as chaotic amplitude control (CAC) [19, 20]. CAC belongs to the class of continuous-state dynamical systems [9, 19, 20, 31], which have been shown to perform significantly better than MCMC methods when properly tuned on hardware that allows parallelization, such as GPUs. The solver has 5 parameters: the non-linear coupling β, rate of change of error variables ξ, gain p, time step Δ, and running time T. More details are given in appendix B.

We measure the computational cost in terms of the number of matrix-vector multiplications K, since these are the most expensive operations per step for this particular solver. The expected time-to-solution is given by equation (2). We train on graphs with number of nodes $N \in \{100, 200, 300\}$, and edge weights chosen uniformly at random from $\{-1, +1\}$.

3.2. Results

To evaluate the performance of the trained model, we use the GNN to predict hyperparameters for a graph not seen during training, then run the solver using those hyperparameters and record the TTS. Again, we are primarily interested in the ability of the model to generalize to larger graphs.

For SK instances, we randomly generate graphs with size $N \in \{400, 500, 700, 800, 900, 1000\}$, and compute the median and standard deviation of the TTS over these instances. The results are shown in Table (1), along with the ground-truth instances for comparison. We see that, even though the model is not trained on graphs of these sizes, it is still able to achieve a good TTS.

For the sparse, 2D toroidal, and scale-free graphs, we evaluate the performance on the GSET instances [5]. For comparison purposes we also include the results in [31], which use hand-tuned hyperparameters for the same solver. The results for graphs of size $N = 800$ are shown in table (2), and for graphs of size up to $N = 2000$ in table (3) in appendix A. We observe that, in the majority of instances, the hyperparameters predicted by the model outperform the hand-tuned ones. This again shows the ability of our method to generalize to much larger graphs.

TTS of Training Data		
N	Median	STD
100	2.08×10^3	2.83×10^3
200	8.59×10^3	8.79×10^3
300	2.11×10^4	8.77×10^4

	TTS with Parameters Predicted by GNN				Hand-tuned
	with post-processing		without post-processing		
N	Median	STD	Median	STD	Median
400	3.51×10^4	1.30×10^5	$\mathbf{2.92 \times 10^4}$	1.73×10^5	1.20×10^5
500	$\mathbf{5.00 \times 10^4}$	2.06×10^5	1.08×10^5	1.87×10^5	2.00×10^5
700	$\mathbf{1.40 \times 10^5}$	7.16×10^5	1.05×10^6	NaN	5.00×10^5
800	$\mathbf{2.20 \times 10^5}$	6.82×10^5	1.23×10^6	NaN	9.00×10^5
900	$\mathbf{2.50 \times 10^5}$	2.98×10^6	5.40×10^6	NaN	1.00×10^6
1000	$\mathbf{3.05 \times 10^5}$	2.01×10^6	NaN	NaN	1.50×10^6

Table 1: Medians and standard deviations of the TTS of 100 fully-connected SK instances of size N. (Top) instances in the training dataset, (bottom) model predictions with and without the post-processing step. For reference, the median TTS found by hand tuning CAC shown in [20] are included.

4. Conclusion and Outlook

We presented our novel method to apply GNNs to combinatorial optimization by predicting hyperparameters for a heuristic solver, and demonstrated that the trained model is able to generalize to much larger graphs than those seen during training.

The framework presented in this work can be generalized to other heuristic solvers, hyperparameter tuning algorithms such derivative-free optimization methods [28, 33], and different GNN architectures such as GATs [39] or GINs [43], providing a flexible approach that can be adapted to various combinatorial optimization tasks.

Instance	N	GNN TTS	Hand-tuned TTS
G1	800	$\mathbf{3.73 \times 10^4}$	9.08×10^4
G2	800	$\mathbf{6.30 \times 10^5}$	9.20×10^5
G3	800	$\mathbf{9.41 \times 10^4}$	1.70×10^5
G4	800	$\mathbf{1.69 \times 10^5}$	2.09×10^5
G5	800	$\mathbf{1.24 \times 10^5}$	2.27×10^5
G6	800	$\mathbf{6.24 \times 10^4}$	1.89×10^5
G7	800	$\mathbf{8.48 \times 10^4}$	5.62×10^5
G8	800	$\mathbf{2.10 \times 10^5}$	4.31×10^5
G9	800	$\mathbf{2.67 \times 10^5}$	4.12×10^5
G10	800	$\mathbf{1.21 \times 10^6}$	1.39×10^6
G11	800	$\mathbf{2.73 \times 10^5}$	3.38×10^5

Instance	N	GNN TTS	Hand-tuned TTS
G12	800	$\mathbf{1.91 \times 10^5}$	2.24×10^5
G13	800	9.97×10^5	$\mathbf{3.91 \times 10^5}$
G14	800	1.99×10^7	$\mathbf{1.73 \times 10^7}$
G15	800	$\mathbf{4.72 \times 10^5}$	5.22×10^5
G16	800	1.03×10^6	$\mathbf{4.94 \times 10^5}$
G17	800	4.27×10^6	$\mathbf{3.09 \times 10^6}$
G18	800	$\mathbf{5.02 \times 10^5}$	5.57×10^5
G19	800	$\mathbf{1.57 \times 10^5}$	1.22×10^6
G20	800	$\mathbf{3.66 \times 10^4}$	1.07×10^5
G21	800	$\mathbf{5.85 \times 10^5}$	3.99×10^6

Table 2: TTS of the solver on the GSET instances of size $N = 800$, using hyperparameters predicted by the GNN vs. those hand-tuned in [31].

References

[1] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012. URL http://jmlr.org/papers/v13/bergstra12a.html.

[2] Stefan Boettcher. Inability of a graph neural network heuristic to outperform greedy algorithms in solving combinatorial optimization problems. *Nature Machine Intelligence*, 5(1):24–25, 2023.

[3] 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.

[4] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, jul 1960. ISSN 0004-5411. doi: 10.1145/321033.321034. URL https://doi.org/10.1145/321033.321034.

[5] Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1), dec 2011. ISSN 0098-3500. doi: 10.1145/2049662.2049663. URL https://doi.org/10.1145/2049662.2049663.

[6] Jian-Ya Ding, Chao Zhang, Lei Shen, Shengyin Li, Bing Wang, Yinghui Xu, and Le Song. Accelerating primal solution findings for mixed integer programs based on solution prediction. In *Proceedings of the aaai conference on artificial intelligence*, volume 34, pages 1452–1459, 2020.

[7] Mária Ercsey-Ravasz and Zoltán Toroczkai. Optimization hardness as transient chaos in an analog approach to constraint satisfaction. *Nature Physics*, 7(12):966–970, 2011. doi: 10.1038/nphys2105. URL https://doi.org/10.1038/nphys2105.

[8] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th*

International Conference on Machine Learning, volume 80 of *Proceedings of Machine Learning Research*, pages 1437–1446. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/falkner18a.html.

[9] Hayato Goto, Kosuke Tatsumura, and Alexander R Dixon. Combinatorial optimization by simulating adiabatic bifurcations in nonlinear hamiltonian systems. *Science advances*, 5(4): eaav2372, 2019.

[10] Hayato Goto, Kotaro Endo, Masaru Suzuki, Yoshisato Sakai, Taro Kanao, Yohei Hamakawa, Ryo Hidaka, Masaya Yamasaki, and Kosuke Tatsumura. High-performance combinatorial optimization based on classical mechanics. *Science Advances*, 7(6):eabe7953, 2021. doi: 10.1126/sciadv.abe7953. URL https://www.science.org/doi/abs/10.1126/sciadv.abe7953.

[11] Emma Hart, Quentin Renau, Kevin Sim, and Mohamad Alissa. Evaluating the robustness of deep-learning algorithm-selection models by evolving adversarial instances. In Michael Affenzeller, Stephan M. Winkler, Anna V. Kononova, Heike Trautmann, Tea Tušar, Penousal Machado, and Thomas Bäck, editors, *Parallel Problem Solving from Nature – PPSN XVIII*, pages 121–136, Cham, 2024. Springer Nature Switzerland. ISBN 978-3-031-70068-2.

[12] Niklas Heim, Tomas Pevny, and Vasek Smidl. Neural power units. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6573–6583. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/48e59000d7dfcf6c1d96ce4a603ed738-Paper.pdf.

[13] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In Carlos A. Coello Coello, editor, *Learning and Intelligent Optimization*, pages 507–523, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-25566-3.

[14] Kirill P Kalinin, George Mourgias-Alexandris, Hitesh Ballani, Natalia G Berloff, James H Clegg, Daniel Cletheroe, Christos Gkantsidis, Istvan Haller, Vassily Lyutsarev, Francesca Parmigiani, et al. Analog iterative machine (aim): using light to solve quadratic optimization problems with mixed variables. *arXiv preprint arXiv:2304.12594*, 2023.

[15] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. ISBN 0-306-30707-3. URL http://dblp.uni-trier.de/db/conf/coco/cocc1972.html#Karp72.

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

[17] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. doi: 10.1126/science.220.4598.671. URL https://www.science.org/doi/abs/10.1126/science.220.4598.671.

[18] Steven M LaValle, Michael S Branicky, and Stephen R Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *The International Journal of Robotics Research*, 23(7-8):673–692, 2004.

[19] Timothée Leleu, Yoshihisa Yamamoto, Peter L. McMahon, and Kazuyuki Aihara. Destabilization of local minima in analog spin systems by correction of amplitude heterogeneity. *Phys. Rev. Lett.*, 122:040607, Feb 2019. doi: 10.1103/PhysRevLett.122.040607. URL https://link.aps.org/doi/10.1103/PhysRevLett.122.040607.

[20] Timothée Leleu, Farad Khoyratee, Timothée Levi, Ryan Hamerly, Takashi Kohno, and Kazuyuki Aihara. Scaling advantage of chaotic amplitude control for high-performance combinatorial optimization. *Communications Physics*, 4(1):266, 2021.

[21] Ang Li, Aleksandra Spyra, Sagi Perel, Valentin Dalibard, Max Jaderberg, Chenjie Gu, David Budden, Tim Harley, and Pramod Gupta. A generalized framework for population based training, 2019. URL https://arxiv.org/abs/1902.01894.

[22] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: a novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.*, 18(1):6765–6816, jan 2017. ISSN 1532-4435.

[23] Andreas Madsen and Alexander Rosenberg Johansen. Neural arithmetic units. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1gNOeHKPS.

[24] J.P. Marques Silva and K.A. Sakallah. Grasp-a new search algorithm for satisfiability. In *Proceedings of International Conference on Computer Aided Design*, pages 220–227, 1996. doi: 10.1109/ICCAD.1996.569607.

[25] Ferenc Molnár, Shubha R. Kharel, Xiaobo Sharon Hu, and Zoltán Toroczkai. Accelerating a continuous-time analog sat solver using gpus. *Computer Physics Communications*, 256: 107469, 2020. ISSN 0010-4655. doi: https://doi.org/10.1016/j.cpc.2020.107469. URL https://www.sciencedirect.com/science/article/pii/S0010465520302204.

[26] Narendra and Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers*, C-26(9):917–922, 1977. doi: 10.1109/TC.1977.1674939.

[27] Dana S. Nau, Vipin Kumar, and Laveen Kanal. General branch and bound, and its relation to a* and ao*. *Artificial Intelligence*, 23(1):29–58, 1984. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(84)90004-3. URL https://www.sciencedirect.com/science/article/pii/0004370284900043.

[28] Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566, 2017.

[29] 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*, volume 33, pages 4731–4738, 2019.

[30] Omri Puny, Heli Ben-Hamu, and Yaron Lipman. Global attention improves graph networks generalization, 2020. URL https://arxiv.org/abs/2006.07846.

[31] Sam Reifenstein, Satoshi Kako, Farad Khoyratee, Timothée Leleu, and Yoshihisa Yamamoto. Coherent ising machines with optical error correction circuits. *Advanced Quantum Technologies*, 4(11):2100077, 2021. doi: https://doi.org/10.1002/qute.202100077. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/qute.202100077.

[32] Sam Reifenstein, Timothee Leleu, Timothy McKenna, Marc Jankowski, Myoung-Gyun Suh, Edwin Ng, Farad Khoyratee, Zoltan Toroczkai, and Yoshihisa Yamamoto. Coherent sat solvers: a tutorial. *Advances in Optics and Photonics*, 15(2):385–441, 2023.

[33] Sam Reifenstein, Timothee Leleu, and Yoshihisa Yamamoto. Dynamic anisotropic smoothing for noisy derivative-free optimization. *arXiv preprint arXiv:2405.01731*, 2024.

[34] Franz Rendl, Giovanni Rinaldi, and Angelika Wiegele. Solving Max-Cut to optimality by intersecting semidefinite and polyhedral relaxations. *Math. Programming*, 121(2):307, 2010.

[35] T. Schoning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, pages 410–414, 1999. doi: 10.1109/SFFCS.1999.814612.

[36] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HJMC_iA5tm.

[37] Robert H Swendsen and Jian-Sheng Wang. Replica monte carlo simulation of spin-glasses. *Physical review letters*, 57(21):2607, 1986.

[38] Andrew Trask, Felix Hill, Scott E Reed, Jack Rae, Chris Dyer, and Phil Blunsom. Neural arithmetic logic units. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/0e64a7b00c83e3d22ce6b3acf2c582b6-Paper.pdf.

[39] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

[40] Amala Vincent and P. Jidesh. An improved hyperparameter optimization framework for automl systems using evolutionary algorithms. *Scientific Reports*, 13, 03 2023. doi: 10.1038/s41598-023-32027-3.

[41] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper_files/paper/2015/file/29921001f2f04bd3baee84a12e98098f-Paper.pdf.

[42] Shuhei Watanabe. Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance, 2023. URL https://arxiv.org/abs/2304.11127.

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

[44] Yoshihisa Yamamoto, Kazuyuki Aihara, Timothee Leleu, Ken-ichi Kawarabayashi, Satoshi Kako, Martin Fejer, Kyo Inoue, and Hiroki Takesue. Coherent ising machines—optical neural networks operating at the quantum limit. *npj Quantum Information*, 3(1):49, 2017. doi: 10.1038/s41534-017-0048-9. URL https://doi.org/10.1038/s41534-017-0048-9.

[45] Zhanguang Zhang, Didier Chételat, Joseph Cotnareanu, Amur Ghose, Wenyi Xiao, Hui-Ling Zhen, Yingxue Zhang, Jianye Hao, Mark Coates, and Mingxuan Yuan. Grass: Combining graph neural networks with expert knowledge for sat solver selection. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, page 6301–6311, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704901. doi: 10.1145/3637528.3671627. URL https://doi.org/10.1145/3637528.3671627.

Appendix A. Supplementary Tables and Figures

Instance	N	GNN TTS	Hand-tuned TTS
G1	800	$\mathbf{3.73 \times 10^4}$	9.08×10^4
G2	800	$\mathbf{6.30 \times 10^5}$	9.20×10^5
G3	800	$\mathbf{9.41 \times 10^4}$	1.70×10^5
G4	800	$\mathbf{1.69 \times 10^5}$	2.09×10^5
G5	800	$\mathbf{1.24 \times 10^5}$	2.27×10^5
G6	800	$\mathbf{6.24 \times 10^4}$	1.89×10^5
G7	800	$\mathbf{8.48 \times 10^4}$	5.62×10^5
G8	800	$\mathbf{2.10 \times 10^5}$	4.31×10^5
G9	800	$\mathbf{2.67 \times 10^5}$	4.12×10^5
G10	800	$\mathbf{1.21 \times 10^6}$	1.39×10^6
G11	800	$\mathbf{2.73 \times 10^5}$	3.38×10^5
G12	800	$\mathbf{1.91 \times 10^5}$	2.24×10^5
G13	800	9.97×10^5	$\mathbf{3.91 \times 10^5}$
G14	800	1.99×10^7	$\mathbf{1.73 \times 10^7}$
G15	800	$\mathbf{4.72 \times 10^5}$	5.22×10^5
G16	800	1.03×10^6	$\mathbf{4.94 \times 10^5}$
G17	800	4.27×10^6	$\mathbf{3.09 \times 10^6}$
G18	800	$\mathbf{5.02 \times 10^5}$	5.57×10^5
G19	800	$\mathbf{1.57 \times 10^5}$	1.22×10^6
G20	800	$\mathbf{3.66 \times 10^4}$	1.07×10^5
G21	800	$\mathbf{5.85 \times 10^5}$	3.99×10^6
G43	1000	$\mathbf{8.03 \times 10^4}$	1.74×10^5
G44	1000	$\mathbf{1.29 \times 10^5}$	2.44×10^5
G45	1000	$\mathbf{4.27 \times 10^5}$	8.81×10^5
G46	1000	$\mathbf{5.42 \times 10^5}$	1.53×10^6

Instance	N	GNN TTS	Hand-tuned TTS
G51	1000	$\mathbf{2.55 \times 10^6}$	3.73×10^6
G52	1000	9.14×10^6	$\mathbf{3.12 \times 10^6}$
G53	1000	2.04×10^7	$\mathbf{1.73 \times 10^7}$
G54	1000	$\mathbf{2.12 \times 10^8}$	2.95×10^8
G22	2000	$\mathbf{2.05 \times 10^6}$	2.52×10^6
G23	2000	NaN	NaN
G24	2000	$\mathbf{1.96 \times 10^6}$	4.79×10^6
G25	2000	$\mathbf{9.80 \times 10^6}$	1.73×10^7
G26	2000	$\mathbf{4.68 \times 10^6}$	1.01×10^7
G27	2000	$\mathbf{3.73 \times 10^5}$	8.36×10^5
G28	2000	$\mathbf{1.26 \times 10^6}$	2.39×10^6
G29	2000	4.84×10^6	$\mathbf{4.16 \times 10^6}$
G30	2000	$\mathbf{3.99 \times 10^7}$	4.21×10^7
G31	2000	$\mathbf{2.16 \times 10^7}$	4.91×10^7
G32	2000	$\mathbf{6.92 \times 10^7}$	7.08×10^7
G33	2000	$\mathbf{2.36 \times 10^8}$	3.07×10^8
G34	2000	3.26×10^7	$\mathbf{2.09 \times 10^7}$
G35	2000	$\mathbf{4.19 \times 10^8}$	NaN
G36	2000	1.66×10^9	$\mathbf{3.68 \times 10^8}$
G37	2000	$\mathbf{1.59 \times 10^9}$	NaN
G38	2000	$\mathbf{3.96 \times 10^7}$	1.08×10^8
G39	2000	4.11×10^6	$\mathbf{4.07 \times 10^6}$
G40	2000	$\mathbf{5.09 \times 10^8}$	1.84×10^9
G41	2000	$\mathbf{2.59 \times 10^6}$	1.71×10^7
G42	2000	2.39×10^7	$\mathbf{1.40 \times 10^7}$

Table 3: TTS of the solver on GSET instances up to size $N = 2000$, using hyperparameters predicted by the GNN vs. those hand-tuned in [31].

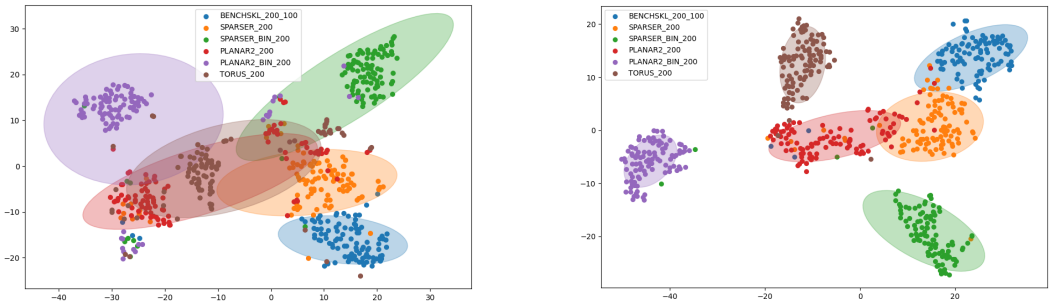

Figure 2: t-SNE of the hyperparameters found by (left) the optimization submodule and (right) the graph neural network. The distinct clusters suggest that the optimal hyperparameters for the heuristic solver can be deduced from the graph structure.

Appendix B. Description of CAC solver

Chaotic amplitude control (CAC) is a method for solving the Ising/Max-Cut problem in which a continuous time dynamical system is integrated numerically [19, 20, 31]. The trajectory of this dynamical system is used to find an approximate solution to the corresponding Ising problem. This method is part of a larger class of algorithms sometimes called "differential solvers" which includes coherent Ising machines (CIM) [44], simulated bifurcation machines (SBM) [10], analog iterative machines (AIM) [14] as well as continuous time dynamical systems to solve the boolean SAT problem [7, 25]. For CAC, the dynamical system is defined by the following set of coupled ODEs:

$$\frac{dx_i(t)}{dt} = (p-1)x_i(t) - x_i(t)^3 + \beta e_i(t) \sum_j J_{ij} x_j(t) \tag{6}$$

$$\frac{de_i(t)}{dt} = \xi e_i(t)(a - x_i(t)^2) \tag{7}$$

where J is the coupling matrix corresponding to the Ising problem being solved and p, β, ξ are hyperparameters of the continuous time dynamical system. Equations (6) and (7) are then numerically integrated for time T using the Euler method with time step Δ which results in the following discrete time dynamical system:

$$x_i(t+\Delta) = x_i(t) + \Delta \cdot \frac{dx_i}{dt} \tag{8}$$

$$e_i(t+\Delta) = e_i(t) + \Delta \cdot \frac{de_i}{dt} \tag{9}$$

This gives us a full set of parameters $\{\Delta, p, \beta, \xi, T\}$, where $K = \frac{T}{\Delta}$ is the total number of time-steps computed per system trajectory. Because of the complicated chaotic dynamics of this system, the values of these parameters need to be selected precisely for each type of problem instance in order for the algorithm to have good performance [20, 31, 33]. Although we use CAC as a test case for this work, this problem of parameter selection is common to many differential solvers and to heuristic Ising solvers in general.

Parameter	Description	Tuning Range
p	rate of linear gain	$[-5, 1]$
β	non-linear coupling strength	$[0.01, 1]$
ξ	rate of change of error variables	$[0.01, 10]$
Δ	time step for Euler integration	$\left[2^{-6}, 2^{-1}\right]$
T	total simulation time	$\left[\frac{N}{10}, 10N\right]$

Table 4: Parameters used in the CAC solver along with the allowed range during hyperparameter optimization.