

---

# Finding Increasingly Large Extremal Graphs with AlphaZero and Tabu Search

---

Abbas Mehrabian<sup>◇\*</sup> Ankit Anand<sup>◇\*</sup> Hyunjik Kim<sup>◇\*</sup>  
Nicolas Sonnerat<sup>◇</sup> Matej Balog<sup>◇</sup> Gheorghe Comanici<sup>◇</sup> Tudor Berariu<sup>○ †</sup>  
Andrew Lee<sup>◇</sup> Anian Ruoss<sup>◇</sup> Anna Bulanova<sup>◇</sup> Daniel Toyama<sup>◇</sup>  
Sam Blackwell<sup>◇</sup> Bernardino Romera Paredes<sup>◇</sup> Petar Veličković<sup>◇</sup>  
Laurent Orseau<sup>◇</sup> Joonkyung Lee<sup>♡</sup> Anurag Murty Naredla<sup>♣</sup>  
Doina Precup<sup>◇ ‡</sup> Adam Zsolt Wagner<sup>♠ ‡</sup>  
◇ Google DeepMind ○ Imperial College London ♣ University of Bonn  
♡ Yonsei University ♠ Worcester Polytechnic Institute

## Abstract

This work studies a central extremal graph theory problem inspired by a 1975 conjecture of Erdős, which aims to find graphs with a given size (number of nodes) that maximize the number of edges without having 3- or 4-cycles. We formulate this problem as a sequential decision-making problem and compare AlphaZero, a neural network-guided tree search, with tabu search, a heuristic local search method. Using either method, by introducing a curriculum—jump-starting the search for larger graphs using good graphs found at smaller sizes—we improve the state-of-the-art lower bounds for several sizes. We also propose a flexible graph-generation environment and a permutation-invariant network architecture for learning to search in the space of graphs.

## 1 Introduction

With the recent advances in neural networks, artificial intelligence (AI) methods have achieved tremendous success in multiple domains like game playing [37], biology [23], mathematics [12], chemistry, and robotics. Mathematics is of particular interest to AI researchers due to its challenging multistep reasoning structure, open-ended problems, and limited data. While automated theorem proving has always been of interest to AI researchers as a reasoning benchmark [2, 5, 25, 26, 33, 35], some recent work have used machine learning to solve research problems across the fields of representation theory, knot theory, and matrix algebra [12, 15, 44].

Many mathematical problems can be modeled as searching for an object or a structure of desired characteristics in an extremely large space. Indeed, automated theorem proving is often modeled as searching for a sequence of operations—a proof—in an ever-growing space of operands with a few operators. Another example is counterexample generation [44], where the object of interest is a counterexample to a particular conjecture or a mathematical construction that improves the bounds for a problem. The recent work of AlphaTensor [15] also relies on neural network-guided tree search to find novel tensor decompositions that result in faster matrix multiplication algorithms.

Inspired by these, we focus on a classical extremal graph theory problem, studied by Erdős [13], which is to find, for any given number of nodes, a graph that maximizes its number of edges but is constrained not to have a 3-cycle or a 4-cycle. While the problem is simple to state, mathematicians

---

\*Equal contribution. Corresponding author: Abbas Mehrabian ([mehrabian@google.com](mailto:mehrabian@google.com)).

†Author was doing an internship at Google DeepMind while this work was done.

‡Equal senior contribution and listed in alphabetical order.

have not found optimal constructions for all sizes: the maximum number of edges is known for up to 53 nodes [22], and lower bounds using local search have been reported for up to 200 nodes [9, 18]. Because there are strong local-search methods, this problem is a simple yet challenging benchmark for learning based search methods. We believe that methods developed for this problem could have wide applications in other fields, ranging from drug discovery to software verification.

In this paper, we use reinforcement learning (RL) and formulate graph generation as a sequential decision making process. In contrast to the graph-generation RL environment used by Wagner [44], which starts from an empty graph and adds edges one by one in a fixed order, we start from an arbitrary graph and add/remove edges in an arbitrary order. This RL environment, called the *edge-flipping* environment, has at least two advantages: (a) we can start from known “good” graphs to find even better graphs, and (b) we can scale to larger sizes by using a curriculum. As our RL agent, we use the state-of-the-art AlphaZero[36] algorithm, which has shown tremendous success in a variety of domains such as tensor decomposition [15], discovering new sorting algorithms [27] and game playing [36].

**Neural network architecture.** Since AlphaZero is guided by a neural network, we also need a representation that aligns with the invariants of the search space. We deal with simple undirected graphs, so graph neural networks naturally come to mind [41]; but we go beyond them and introduce a novel representation called the *pairformer*: unlike traditional graph neural networks, which pass messages between nodes, pairformers pass messages between *pairs of nodes*. Pairformers burden us with additional computational cost but are significantly better at detecting cycles.

**Curriculum learning.** When searching over all graphs, one challenge is that the number of graphs with a given number of nodes  $n$  increases exponentially with  $n$ ; thus, finding optimal graphs becomes significantly harder as  $n$  grows. Interestingly, known optimal graphs for this problem have a substructure property: in many cases, optimal graphs of a given size are near-subgraphs of optimal graphs of larger sizes (see, e.g., [7, Theorem 3]). Thus, finding near-optimal graphs for smaller  $n$  can serve as a stepping stone to find good graphs for larger values of  $n$ . This property can be used to construct a curriculum: start from discovered graphs of a given size, generate novel solutions of a larger size, and repeat. Our edge-flipping environment provides the flexibility to start from any graph and add or drop edges arbitrarily, so we are not restricted to supergraphs of the starting graph and can reach all graphs of that size. Deploying these ideas in AlphaZero, we improve the lower bounds for sizes 64 to 136.

Curriculum learning [38] is a widely-used method in RL, especially for solving hard exploration problems in many domains, including robotics [4] and automated theorem proving [5] as well as solving the Rubik’s cube and other difficult puzzles [3, 31]. It should be noted that the term curriculum learning has been used with different contexts in machine learning literature, for this paper, we use the term *curriculum learning* for solving a problem on small size first and using the solution of smaller problem to solve the same problem with larger size.

**Local search with curriculum.** The substructure property can also enhance other types of search. We develop an incremental version of tabu search, a known local search method [19], where the initial graph for each size is sampled from a previously-discovered “good” graph of smaller size. This algorithm also improves over the state of the art. Our ablation shows that both search strategies improve significantly by using a curriculum which leverages the substructure property.

**Summary of contributions.** We introduce a challenging benchmark for learning-to-search in large state spaces, inspired by an open problem in extremal graph theory, whose best solutions thus far are achieved by local search. We formulate graph generation as an edge-flipping environment and introduce the novel representation pairformer, which is well-suited for detecting cycles in undirected graphs. We introduce the use of a curriculum to local search methods as well as AlphaZero, and show that it is a key ingredient for improving the results on this extremal graph theory problem. Lastly, we improve the lower bounds for the problem for all graph sizes from 64 to 134, and we release these graphs to the research community to aid further research. The graphs can be found at [https://storage.googleapis.com/gdm\\_girth5\\_graphs/girth5\\_graphs.zip](https://storage.googleapis.com/gdm_girth5_graphs/girth5_graphs.zip).

## 2 Preliminaries

We consider simple and undirected graphs only. Let  $f(n)$  denote the maximum number of edges possible in an  $n$ -node graph without a 3-cycle or a 4-cycle (see Table 1 in Appendix for values of  $f(n)$  for small  $n$ ). Erdős [13] conjectured that  $\lim_{n \rightarrow \infty} \frac{f(n)}{n\sqrt{n}} = \frac{1}{2\sqrt{2}}$ . This conjecture has remained open since 1975 (see Appendix B for more background). The tightest bounds are due to Garnick et al. [18], who proved

$$\frac{1}{2\sqrt{2}} \leq \lim_{n \rightarrow \infty} \frac{f(n)}{n\sqrt{n}} \leq \frac{1}{2}. \quad (1)$$

Motivated by this conjecture, we want to estimate the value of  $f(n)$  for specific values of  $n$ . It is known that  $f(n) \leq n\sqrt{n-1}/2$  for all  $n$  [18]. The exact value of  $f(n)$  is known when  $n \leq 53$  [22], and constructive lower bounds have been reported for all  $n \leq 200$  [9, 18]. Our goal is to improve these lower bounds for  $54 \leq n \leq 200$ . Hence, we want to find  $n$ -node graphs without 3-cycles or 4-cycles that have as many edges as possible.

The *size* of a graph is its number of nodes. For any graph  $G$ , we denote its number of edges, 3-cycles, and 4-cycles by  $e(G)$ ,  $\Delta(G)$ , and  $\square(G)$ , respectively. We say that a graph  $G$  is *feasible* if it has no 3-cycles and no 4-cycles. The *score* of a graph  $G$  is defined as

$$s(G) := e(G) - \Delta(G) - \square(G). \quad (2)$$

The following lemma (proof in Appendix C) implies that, for any given number of nodes  $n$ , proving lower bounds for  $f(n)$  is equivalent to maximizing the score over all  $n$ -node graphs.

**Lemma 1.** *For any  $n$ -node graph  $G$ , we have  $s(G) \leq f(n)$ ; and there exists at least one  $n$ -node feasible graph for which equality holds.*

In light of Lemma 1, we can formulate the problem of maximizing  $f(n)$  in two ways: we can maximize  $e(G)$  over *feasible*  $n$ -node graphs or maximize  $s(G)$  over *all*  $n$ -node graphs. The two formulations have the same optimal value, but their search space differs. The first formulation has a smaller search space, but the second one, which we use, allows us to define a convenient *neighborhood function*, which helps our algorithms navigate the space of graphs more smoothly.

**Definition 1** (Flipping). *Let  $G$  be a graph and let  $u$  and  $v$  be two of its nodes. If  $uv$  is an edge in  $G$ , then  $G \oplus uv$  is obtained by removing the edge  $uv$  from  $G$ ; otherwise,  $G \oplus uv$  is obtained by adding the edge  $uv$ . In either case, we say  $G'$  is obtained by flipping  $uv$ .*

The flipping operation has two desirable properties: first, any  $n$ -node graph assumes exactly  $\binom{n}{2}$  flips, a technical convenience for RL agents’ action space; second, any  $n$ -node graph can be reached from any other  $n$ -node graph by doing up to  $\binom{n}{2}$  many flips; that is, there are no “dead ends.”

## 3 Methods

A key observation about our problem is that, in many cases, the optimal graph for a given size is nearly a subgraph of an optimal graph for a larger size. For example, by [7, Theorem 3], all the optimal graphs for sizes 40, 45, 47, 48, 49 are subgraphs of the optimal graph for size 50. While this is not strictly true for all the sizes, it can be used as a heuristic to guide the search. As all the optimal graphs up to size 52 are known [30], we use high-scoring graphs of smaller sizes (garnished with a suitable number of isolated nodes) as the initial graph and iterate. Hence if the results for smaller sizes improve, this hopefully leads to subsequent improvements for larger sizes as well. This not only exploits the approximate substructure property for the problem but is also motivated by the well-known idea of curriculum learning in machine learning [8].

We study two main methods in this work: AlphaZero, which exemplifies reinforcement learning, and tabu search, which exemplifies local search.

### 3.1 AlphaZero

We define graph generation as an edge-flipping RL environment, where the state space consists of all possible graphs of a given size  $n$ , the actions consist of all  $\binom{n}{2}$  pairs of nodes, the transitions are

adding or removing an edge in the current graph, and the reward equals the change in the score,  $s(G)$ , after taking an action.

The AlphaZero agent starts from a given graph and builds a tree search by balancing exploration and exploitation using the UCB rule [24]. The tree search is guided by two neural networks, the policy and value networks. These networks share a common torso with a *pairformer* architecture, which inputs the adjacency matrix and outputs a graph embedding. This graph embedding is transformed by the policy and value heads to a probability distribution over actions and the value of a state, respectively.

We implement two versions of AlphaZero: one starting from the empty graph and one starting from graphs of smaller sizes that have a good score. The latter leads to a natural curriculum and improves the performance significantly. The agents is trained using a distributed infrastructure, where several actors operate parallelly and feed the data to learner, which in turn feeds the updated parameters to the actors. See Appendix D for details.

### 3.2 Tabu search

Local search has been the main scheme to prove lower bounds for  $f(n)$ . We implement a widely used local search method based on tabu search. We observe that tabu search also benefits from starting from good graphs of smaller sizes, so we create an incremental local search framework and show that this idea significantly improves the performance of tabu search. See Appendix F for details.

## 4 Experiments and results

We compare five methods: tabu search starting from the empty graph; incremental tabu search, which uses a curriculum to use high-scoring graphs found at each size as the starting point for larger sizes (explained in Appendix F); AlphaZero starting from the empty graph; incremental AlphaZero, which uses a curriculum (explained in Appendix D); and Wagner’s cross-entropy method [44], the first machine-learning method to find counterexamples for mathematical conjectures (see Appendix G). For AlphaZero, we also perform ablations on the choice of network representation (see Appendix E). Since  $f(n) = \Theta(n\sqrt{n})$  (see (1)) we have normalized the scores by  $n\sqrt{n}$  in the plots.

**Comparison with the state-of-the-art lower bounds.** As Figure 1 shows, incremental tabu search improves the state-of-the-art lower bounds [1, 17, 18, 30] when  $n \in \{64, \dots, 134\} \cup \{138, \dots, 160\} \cup \{176, \dots, 186\} \cup \{188, \dots, 190\}$  (see Appendix A for a concrete example).<sup>4</sup> In Table 2, we have listed the lower bounds achieved by incremental tabu search for  $n = 1, 2, \dots, 200$ . AlphaZero with curriculum also improves over the state-of-the-art lower bounds on many sizes, and is exactly on par with incremental tabu search on all sizes between 54 to 100, except  $n = 56, 57, 64, 66, 77$ , and 96, where incremental tabu search leads by one edge. We observe that Wagner’s cross entropy method [44] performs worse than both tabu search and AlphaZero. See details in Appendix G.

**Benefits of using a curriculum.** Without curriculum, the agent must build graphs of a given size starting from the empty graph, while with curriculum, the agent starts from a previously-found graph of a smaller size and flips some edges to obtain a graph of the desired size. Figure 2 (Left) illustrates, for sizes 54 to 100, the benefit of using a curriculum for tabu search, which increases significantly as the number of nodes increases. Figure 2 (Right) shows a similar plot for AlphaZero: AlphaZero without curriculum matches AlphaZero with curriculum up to size around 30 but deteriorates afterwards. We believe the reasons are large episode lengths and the difficulties of exploration and credit assignment. We conclude that using a curriculum is a vital ingredient for applying reinforcement learning to this problem, and presumably for any optimization problem with a substructure property and a huge state space.

<sup>4</sup>We have not compared against the lower bounds reported in [9], as neither the graphs achieving those bounds nor the method for generating them are presented in [9]; still, incremental tabu search improves over the lower bounds reported in [9] for  $n \in \{64, \dots, 76\}$ .

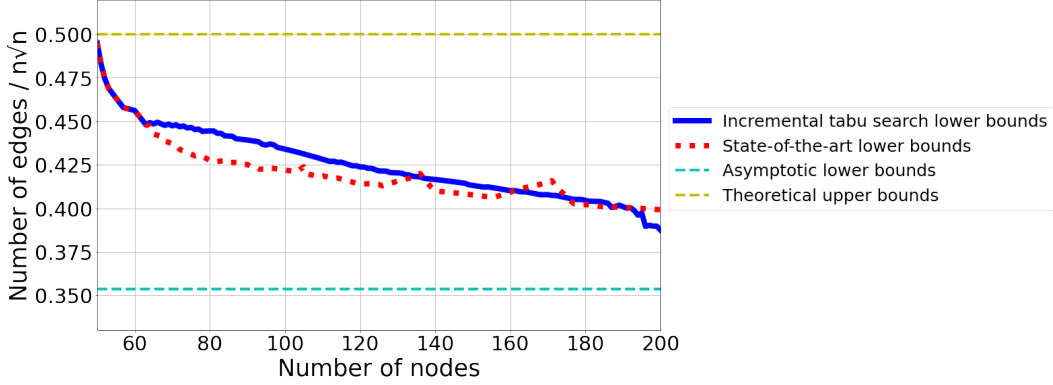


Figure 1: Normalized scores, given by  $\frac{\text{number of edges}}{n\sqrt{n}}$ , are plotted versus size,  $n$ . AlphaZero with curriculum (not plotted) achieves the same score as incremental tabu search for 41 of the sizes from 54 to 100. Erdős conjectured that both the red and blue curves converge to the cyan horizontal line as  $n \rightarrow \infty$ . State-of-the-art lower bounds are from [1, 17, 30, 18], theoretical upper bounds from [18].

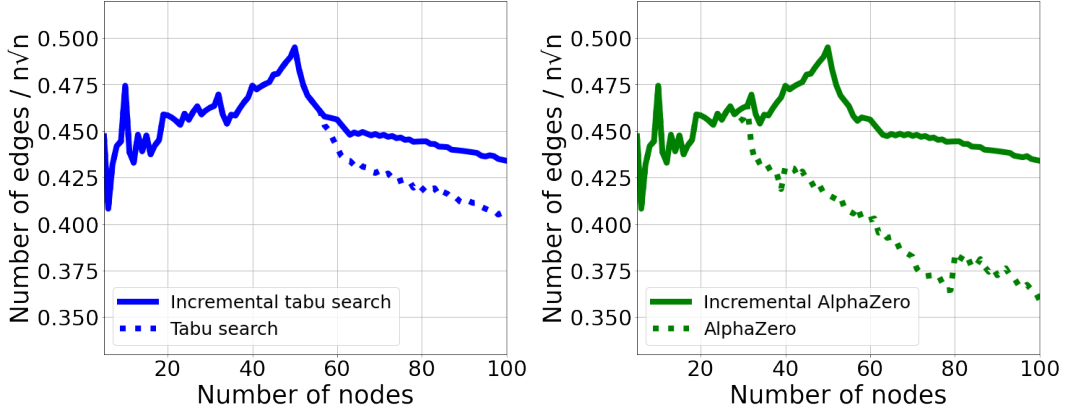


Figure 2: Left: incremental tabu search, which uses a curriculum, performs increasingly better than tabu search without curriculum, for larger problem sizes. Right: adding a curriculum improves the performance of AlphaZero significantly, especially on larger sizes.

## 5 Discussion

We studied a challenging learning-to-search benchmark inspired by an open problem in extremal graph theory. We compared a neural network-guided Monte Carlo tree search with tabu search, and observed that using a curriculum is crucial for improving the results but adding learning did not improve the results (a similar phenomenon was observed for another extremal graph theory problem, see [32]). This may be due to the problem having lots of local optima, making it hard to explore. Also, for some sizes (e.g., 40, 45, 47, 48, and 49), there is only one  $n$ -node feasible graph with score  $f(n)$  [7, Theorem 3]. To give another example, during our experiments, there was a size,  $n = 96$ , for which only one of our runs found a score of 411, and all other runs found smaller scores. In contrast to problems on which RL has improved the state of the art (e.g., the tensor decomposition problem [15]), which do not have strong local search baselines, for this problem, natural, fast, and strong local search algorithms exist. Crucially, tabu search explores much faster than AlphaZero: in a given amount of time, the former sees and evaluates orders of magnitude more graphs than the latter. Finally, in contrast to a typical RL problem, where one aims to maximize the *expected return* in a non-deterministic environment, in this problem we want to maximize the *best-case* return in a deterministic environment. In other words, we need only find a good solution once. This makes one wonder whether RL is the right approach for this problem.

## Acknowledgments and Disclosure of Funding

We are grateful to Eser Aygün for advising us throughout this project and his helpful comments on the first draft of the paper. We also thank Brendan McKay for releasing the best-known graphs up to size 64, which helped us jump-start our incremental tabu search algorithm.

We appreciate several useful discussions with David Applegate, Charles Blundell, Alex Davies, Matthew Fahrbach, Alhussein Fawzi, Michael Figurnov, David Garnick, Xavier Glorot, Harris Kwong, Felix Lazebnik, Shibl Mourad, Sébastien Racaniere, Tara Thomas, and Theophane Weber.

Joonkyung Lee is supported by Samsung STF Grant SSTF-BA2201-02.

## References

- [1] E. Abajo, C. Balbuena, and A. Diánez. New families of graphs without short cycles and large size. *Discrete Appl. Math.*, 158(11):1127–1135, 2010. ISSN 0166-218X. doi: 10.1016/j.dam.2010.03.007.
- [2] I. Abdelaziz, M. Crouse, B. Makni, V. Austel, C. Cornelio, S. Ikbali, P. Kapanipathi, N. Makondo, K. Srinivas, M. Witbrock, and A. Fokoue. Learning to guide a saturation-based theorem prover. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 2022. ISSN 1939-3539.
- [3] Forest Agostinelli, Stephen McAleer, Alexander Shmakov, and Pierre Baldi. Solving the Rubik’s cube with deep reinforcement learning and search. *Nature Machine Intelligence*, 1, 07 2019.
- [4] İlge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving Rubik’s cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.
- [5] Eser Aygün, Ankit Anand, Laurent Orseau, Xavier Glorot, Stephen M Mcaleer, Vlad Firoiu, Lei M Zhang, Doina Precup, and Shibl Mourad. Proving theorems using incremental learning and hindsight experience replay. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 1198–1210. PMLR, 2022.
- [6] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.
- [7] Jörgen Backelin. Sizes of the extremal girth 5 graphs of orders from 40 to 49. *arXiv preprint, arXiv:1511.08128v1 [math.CO]*, 2015.
- [8] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.
- [9] Novi H. Bong. Some new upper bounds of  $ex(n; C_3, C_4)$ . *AKCE International Journal of Graphs and Combinatorics*, 14(3):251–260, 2017. ISSN 0972-8600. doi: <https://doi.org/10.1016/j.akcej.2017.03.006>. URL <https://www.sciencedirect.com/science/article/pii/S0972860016300676>.
- [10] W. G. Brown. On graphs that do not contain a Thomsen graph. *Can. Math. Bull.*, 9:281–285, 1966. ISSN 0008-4395. doi: 10.4153/CMB-1966-036-2.
- [11] Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In H. Jaap van den Herik, Paolo Ciancarini, and H. H. L. M. (Jeroen) Donkers, editors, *Computers and Games*, pages 72–83, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-75538-8.
- [12] Alex Davies, Petar Veličković, Lars Buesing, Sam Blackwell, Daniel Zheng, Nenad Tomašev, Richard Tanburn, Peter Battaglia, Charles Blundell, András Juhász, Marc Lackenby, Geordie Williamson, Demis Hassabis, and Pushmeet Kohli. Advancing mathematics by guiding human intuition with AI. *Nature*, 600(7887):70–74, 2021.



- [13] Paul Erdős. Some recent progress on extremal problems in graph theory. *Congr. Numer.*, 14: 3–14, 1975.
- [14] Paul Erdős, Alfréd Rényi, and Vera T. Sós. On a problem of graph theory. *Stud. Sci. Math. Hung.*, 1:215–235, 1966. ISSN 0081-6906.
- [15] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatin, Alexander Novikov, Francisco J. R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, David Silver, Demis Hassabis, and Pushmeet Kohli. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610:47–53, 2022.
- [16] Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. In Sergey Levine, Vincent Vanhoucke, and Ken Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 482–495. PMLR, 13–15 Nov 2017. URL <https://proceedings.mlr.press/v78/florensa17a.html>.
- [17] David K Garnick. Extremal graphs without three-cycles or four-cycles. Personal communication, 2023.
- [18] David K Garnick, YH Harris Kwong, and Felix Lazebnik. Extremal graphs without three-cycles or four-cycles. *Journal of Graph Theory*, 17(5):633–645, 1993.
- [19] Fred Glover. Tabu search. I. *ORSA J. Comput.*, 1(3):190–206, 1989. ISSN 0899-1499. doi: 10.1287/ijoc.1.3.190.
- [20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [21] Ionel-Alexandru Hosu and Traian Rebedea. Playing Atari games with deep reinforcement learning and human checkpoint replay. *arXiv preprint arXiv:1607.05077*, 2016.
- [22] OEIS Foundation Inc. Maximal number of edges in  $n$ -node graph of girth at least 5. Entry A006856 in the on-line encyclopedia of integer sequences. <https://oeis.org/A006856>, 2023. Accessed: September 13, 2023.
- [23] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, 2021.
- [24] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, pages 282–293, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-46056-5.
- [25] Laura Kovács and Andrei Voronkov. First-order theorem proving and vampire. In *International Conference on Computer Aided Verification*, pages 1–35. Springer, 2013.
- [26] Guillaume Lample, Timothee Lacroix, Marie-Anne Lachaux, Aurelien Rodriguez, Amaury Hayat, Thibaut Lavril, Gabriel Ebner, and Xavier Martinet. Hypertree proof search for neural theorem proving. *Advances in Neural Information Processing Systems*, 35:26337–26349, 2022.
- [27] Daniel J Mankowitz, Andrea Michi, Anton Zhernov, Marco Gelmi, Marco Selvi, Cosmin Paduraru, Edouard Leurent, Shariq Iqbal, Jean-Baptiste Lespiau, Alex Ahern, Thomas Köppe, Kevin Millikin, Stephen Gaffney, Sophie Elster, Jackson Broshear, Chris Gamble, Kieran Milan, Robert Tung, Minjae Hwang, Taylan Cemgil, Mohammadamin Barekatin, Yujia Li, Amol Mandhane, Thomas Hubert, Julian Schrittwieser, Demis Hassabis, Pushmeet Kohli, Martin Riedmiller, Oriol Vinyals, and David Silver. Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, 618(7964):257–263, 2023.

- [28] Willem Mantel. Vraagstuk xxviii. *Wiskundige Opgaven met de Oplossingen*, 10(2):60–61, 1907.
- [29] Brendan McKay. Description of graph6, sparse6 and digraph6 encodings. <https://users.cecs.anu.edu.au/~bdm/data/formats.txt>, 2022. Accessed: September 13, 2023.
- [30] Brendan McKay. Extremal graphs and Turan numbers. <https://users.cecs.anu.edu.au/~bdm/data/extremal.html>, 2023. Accessed on 7 August 2023.
- [31] Laurent Orseau, Marcus Hutter, and Levi H. S. Lelis. Levin tree search with context models. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 5622–5630. International Joint Conferences on Artificial Intelligence Organization, 2023.
- [32] Olaf Parczyk, Sebastian Pokutta, Christoph Spiegel, and Tibor Szabó. New Ramsey multiplicity bounds and search heuristics. *arXiv preprint, arXiv:2206.04036v2 [math.CO]*, 2023. Conference version in Proceedings of the AAAI Conference on Artificial Intelligence, 2023.
- [33] Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020.
- [34] Tim Salimans and Richard Chen. Learning montezuma’s revenge from a single demonstration. *arXiv preprint arXiv:1812.03381*, 2018.
- [35] Stephan Schulz. E—a brainiac theorem prover. *AI Communications*, 15(2, 3):111–126, 2002.
- [36] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 2017.
- [37] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dhharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362:1140–1144, 2018.
- [38] Petru Soviany, Radu Tudor Ionescu, Paolo Rota, and Nicu Sebe. Curriculum learning: A survey. *Int. J. Comput. Vision*, 130(6):1526–1565, jun 2022. ISSN 0920-5691. doi: 10.1007/s11263-022-01611-x. URL <https://doi.org/10.1007/s11263-022-01611-x>.
- [39] Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. Monte Carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, 56(3):2497–2562, 2023.
- [40] Pál Turán. On an extremal problem in graph theory. *Mat. Fiz. Lapok*, 48:436–452, 1941.
- [41] Petar Veličković. Everything is connected: Graph neural networks. *arXiv preprint arXiv:2301.08210*, 2023.
- [42] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.
- [43] Petar Veličković, Lars Buesing, Matthew Overlan, Razvan Pascanu, Oriol Vinyals, and Charles Blundell. Pointer graph networks. *Advances in Neural Information Processing Systems*, 33: 2232–2244, 2020.
- [44] Adam Zsolt Wagner. Constructions in combinatorics via neural networks. *Preprint, arXiv:2104.14516 [math.CO]*, 2021.



Table 1: The values of  $f(n)$  for  $n = 1, 2, \dots, 9$ .





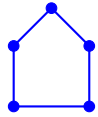

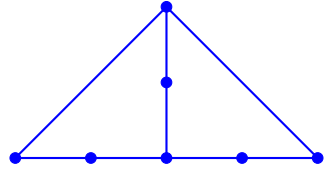
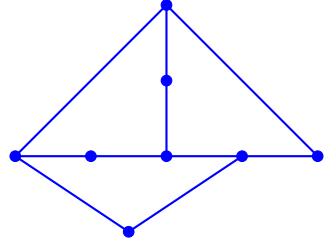
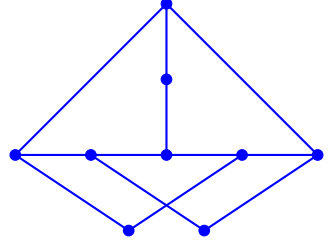
$n$	$f(n)$	all $n$ -node feasible graphs with $f(n)$ edges
1	0	
2	1	
3	2	
4	3	
5	5	
6	6	
7	8	
8	10	
9	12	

Table 2: The maximum number of edges of graphs with no 3- or 4-cycles found by incremental tabu search and the number of non-isomorphic graphs found for each size.

Num. nodes	Num. edges	Num. graphs	Num. nodes	Num. edges	Num. graphs
1	0	1	51	176	7
2	1	1	52	178	102
3	2	1	53	181	402
4	3	2	54	185	5
5	5	1	55	189	5
6	6	2	56	193	11
7	8	1	57	197	4
8	10	1	58	202	1
9	12	1	59	207	2
10	15	1	60	212	2
11	16	3	61	216	16
12	18	7	62	220	85
13	21	1	63	224	2662
14	23	4	64	230	2
15	26	1	65	235	74
16	28	22	66	241	1
17	31	14	67	246	43
18	34	15	68	251	979
19	38	1	69	257	85
20	41	1	70	262	1575
21	44	2	71	268	66
22	47	3	72	273	694
23	50	7	73	279	4
24	54	1	74	284	172
25	57	6	75	290	12
26	61	2	76	295	1298
27	65	1	77	301	1
28	68	4	78	306	548
29	72	1	79	312	39
30	76	1	80	318	25
31	80	2	81	324	11
32	85	1	82	329	673
33	87	12	83	335	22
34	90	230	84	340	375
35	95	5	85	346	9
36	99	34	86	352	4
37	104	6	87	357	584
38	109	2	88	363	288
39	114	1	89	369	50
40	120	1	90	375	8
41	124	1	91	381	2
42	129	1	92	387	10
43	134	1	93	393	1
44	139	2	94	398	1014
45	145	1	95	404	605
46	150	2	96	411	1
47	156	1	97	417	4
48	162	1	98	422	819
49	168	1	99	428	161
50	175	1	100	434	49

Num. nodes	Num. edges	Num. graphs	Num. nodes	Num. edges	Num. graphs
101	440	15	151	766	8
102	446	3	152	773	5
103	452	5	153	780	2
104	458	8	154	788	3
105	464	10	155	795	14
106	470	6	156	802	1
107	476	9	157	809	10
108	482	7	158	816	18
109	488	14	159	823	27
110	494	100	160	830	12
111	500	119	161	838	2
112	506	105	162	845	16
113	513	4	163	852	24
114	519	30	164	860	7
115	526	1	165	867	5
116	532	3	166	874	4
117	538	34	167	881	20
118	544	27	168	888	32
119	551	5	169	896	3
120	557	33	170	904	4
121	564	1	171	911	19
122	570	22	172	919	1
123	577	1	173	926	7
124	583	18	174	933	21
125	589	67	175	940	4
126	596	3	176	947	4
127	603	1	177	954	46
128	609	3	178	962	4
129	616	34	179	970	1
130	623	15	180	977	2
131	630	1	181	984	12
132	636	41	182	992	6
133	643	6	183	1000	8
134	649	40	184	1008	6
135	656	3	185	1015	1
136	663	1	186	1022	2
137	669	54	187	1024	13
138	676	32	188	1034	1
139	683	17	189	1044	1
140	690	18	190	1050	1
141	697	9	191	1056	1
142	704	7	192	1065	1
143	711	4	193	1069	4
144	718	3	194	1070	1
145	725	3	195	1082	1
146	732	12	196	1069	5
147	739	4	197	1079	1
148	746	2	198	1086	7
149	752	9	199	1094	6
150	759	7	200	1096	2

## A An example

As a concrete example of our results, we provide the sparse6 representation [29] of one graph on 64 nodes with 230 edges without 3- or 4-cycles we found via incremental tabu search (the best published bound is 229 [30]):

```
b'>>sparse6<<:~?@?_0GoIO?AoxKSKFD0@SQKKbBXCGECAo^Kwd0j?_ |E1DHeG0grKFoOgcYMjFH?cpLLFh` [y^_Bq[0WQIEGwwaTNOpX`
?`ba@xH@DHEr{WQSIeSCCUONghp?mWRQ@0seTJDw_keSLfRs?GFRiWP?iZMHGWwo\POPO{aSKP@?wsc_ERiMEZSIdeSWU``p_s_XbFeRMM
SJebaDctecaaLGrhf tat [nXtRiX]oYLWH[qZVLw_c_U0mhG{mgwKU[CETLHTI1fCDHTjDjBFDAp| |\n'
```

## B Problem background

Let  $G$  be a simple, undirected  $n$ -node graph that has no 3-cycles. What is the maximum number of edges that  $G$  can have? Mantel [28] proved that the answer is precisely  $\lfloor n^2/4 \rfloor$ , initiating the field of extremal graph theory. Turán [40] generalized this result to cliques and found, for any  $k$ , the maximum number of edges that an  $n$ -node graph without  $k$ -cliques can have.

Generally, for a set  $\mathcal{H}$  of graphs, let  $\text{ex}(n, \mathcal{H})$  denote the maximum number of edges in an  $n$ -node graph that does not contain any member of  $\mathcal{H}$  as a subgraph (the symbol  $\text{ex}$  stands for “extremal”). Calculating  $\text{ex}(n, \mathcal{H})$  for various graph classes  $\mathcal{H}$  is a central problem in extremal graph theory. In this paper, we study

$$f(n) := \text{ex}(n, \{C_3, C_4\}). \quad (3)$$

We know  $\text{ex}(n, \{C_3\}) = \lfloor n^2/4 \rfloor$  by Mantel’s theorem and  $\lim_{n \rightarrow \infty} \text{ex}(n, \{C_4\})/(n\sqrt{n}) = 1/2$  by [10, 14], but no formula has been found for  $f(n)$ , and even its asymptotic behavior is not understood well. The tightest bounds are due to Garnick et al. [18], who proved

$$\frac{1}{2\sqrt{2}} \leq \lim_{n \rightarrow \infty} \frac{f(n)}{n\sqrt{n}} \leq \frac{1}{2}.$$

Erdős [13] suggested that the left inequality is indeed an equality. This conjecture has remained open since 1975.

## C Proof of Lemma 1

Let  $G$  be any  $n$ -node graph and apply the following procedure: while  $G$  has at least one 3-cycle or 4-cycle, choose any such cycle arbitrarily and delete one of its edges; repeat until no 3- or 4-cycles remain. Denote the resulting graph by  $G'$ . Since  $\Delta(G') = \square(G') = 0$ , we have  $s(G') = e(G') \leq f(n)$ . Also, for each 3-cycle or 4-cycle of  $G$ , we have deleted at most one edge during this procedure, hence  $s(G') = e(G') \geq e(G) - \Delta(G) - \square(G) = s(G)$ . Therefore, for all  $n$ -node graphs  $G$ ,  $s(G) \leq s(G') \leq f(n)$ , proving the first part of the lemma. The second part of the lemma follows from the definition of  $f(n)$ : see (3).

## D Incremental AlphaZero

AlphaZero is a reinforcement learning algorithm that demonstrated superhuman performance on Go, through self-play, without using any human knowledge [36]. It was then adapted to show superhuman performance in other games such as chess and shogi [37]. Recently, a version of AlphaZero was adapted to find faster algorithms for matrix multiplication. This new model, named AlphaTensor, improved Strassen’s matrix-multiplication algorithm for some sizes for the first time after 50 years [15]. The AlphaZero algorithm combines Monte Carlo Tree Search (MCTS), a heuristic search algorithm, with deep neural networks to represent the state space, e.g., a chessboard position. In our application of AlphaZero, each state is a simple undirected graph and each action is adding or removing an edge.

### D.1 Graph generation as an RL environment

We define *graph generation* as a sequential decision making process, where we start from an  $n$ -node graph  $G$  and, at each step, modify it by adding or removing an edge  $e$  to obtain  $G' = G \oplus e$ . More formally, graph generation is a deterministic finite-horizon Markov Decision Process (MDP)

$\mathcal{M} = \{\mathcal{S}, \mathcal{A}, R, \mathcal{T}, H\}$ , where the state space,  $\mathcal{S}$ , consists of all simple undirected graphs of size  $n$ ; the action space,  $\mathcal{A} := \{(i, j) : 1 \leq i < j \leq n\}$ , consists of all edges of the complete graph of size  $n$ ; the one-step reward function (defined below) is  $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ ; the deterministic transition function,  $\mathcal{T}$ , is defined as  $\mathcal{T}(G, e) := G \oplus e$ ; and the horizon,  $H$ , denotes the number of steps in each episode.

We define the reward function as  $R(G, e) := s(G \oplus e) - s(G)$ , i.e., the reward equals the change in the score after taking the action. We call this *the telescopic reward*, as the rewards accumulated over time form a telescoping series, making the episode return equal to  $s(G_H) - s(G_0)$ . Note that  $s(G_H)$  is precisely the objective value that we want to maximize. (Often in RL, a discount factor is introduced, and later rewards are discounted when computing the return; but we did not introduce a discount factor here, as then the return would have been different from the actual objective function.) In our experiments, we found that the telescoping reward performs much better than the *non-telescoping* reward, where the reward is given at the end of each episode and equals  $s(G_H)$ .

This *edge-flipping environment* provides more flexibility than environments in which the graph is built, for instance, by deciding about the edges one by one in a fixed order [44]. An advantage of the edge-flipping environment is that every action is reversible: in this MDP, any  $n$ -node graph can be reached from any other  $n$ -node graph. This property is particularly useful when using a curriculum, as one can use *state resetting* [16, 21, 34] to warm up exploration from high-scoring initial graphs; e.g., start from high quality graphs of size  $n - k$  (garnished with  $k$  isolated nodes) to build the desired graph of size  $n$ . On the other hand, reversibility can cause learning instability: since there is no termination action, an agent could end up flipping one of the edges indefinitely. To avoid such issues, we set a fixed horizon length.

## D.2 Monte Carlo tree search

The edge-flipping environment is a deterministic MDP, where both the transition matrix and the reward function are fully known and deterministic, thus most search and planning algorithms are applicable. In this work, we use Monte Carlo Tree Search (MCTS) [11], which has had great success in large state and action spaces [39]. MCTS builds a finite tree rooted at the current state and, based on the statistics gathered from the neighboring states, selects the next action. Many successful works using MCTS use some variant of the upper confidence bound rule [24] to balance exploration and exploitation when expanding the tree. While traditional approaches used Monte Carlo rollouts to estimate the value of a leaf state, in the last decade this has largely been replaced by a neural network, called the *value network*. Another neural network, called the *policy network*, determines which child to expand next. Often, the policy and value networks share the same first few layers. (They have the same latent representation, or torso, but they have different heads.) In AlphaZero, both the policy and value networks are trained using previously observed trajectories—see [37] for details.

For updating the value of a state—which is a node in the MCTS tree—standard MCTS expands the node and uses the average value of the children. Since we want to maximize the best-case return rather than the expected return, it may appear more suitable to use the maximum value of the children to update the value of the node. We attempted this approach but it did not yield improvements.

One common issue with AlphaZero is encouraging it to diversely explore the space of possible trajectories. We attempted a few ideas to achieve this, such as increasing UCB exploration parameter and also for each trajectory, if same graph ( $s_i$ ) is encountered which has been seen in the previous timesteps ( $t < i$ ) within the trajectory, we discourage this behavior by giving a small negative reward but none of these approaches improved the result on top of starting from good graphs of smaller size.

## D.3 Network representation

To find good representation for this problem of avoiding short cycles, we studied different architectures in the supervised problem of cycle detection, including resnets [20], pointer graph networks [43], graph attention networks [42] and a novel architecture called *pairformers*, which we describe below. We studied binary short-cycle detection tasks at node and edge levels (whether a node or an edge is part of a short cycle) as well as graph level (whether a graph contains a short cycle) level. Pairformers worked best hence we used them in the RL setting as well.

The pairformer is a simplified version of the Evoformer used in AlphaFold [23]. Each Evoformer block has two branches of computation: one processes the multiple sequence alignment (MSA)

representation and the other one processes the pair representation. The pairformer only uses the pair representation branch, which processes per-edge features and has shape  $(n, n, c)$ . We use  $c=64$  for our implementations. Within the pair representation branch, each pairformer block is composed of the triangle self-attention blocks (row-wise multihead self-attention followed by column-wise multihead self-attention) followed by fully-connected layers with LayerNorm [6]. (The triangle multiplicative updates in the original Evoformer are unused.) A key difference with standard graph neural networks is that instead of only having features for existing edges, the pairformer has features for all  $\binom{n}{2}$  pairs of nodes, whether they correspond to existing edges or not. We believe that considering non-existing edges is crucial for the pairformer to inform the policy to decide about adding new edges to the graph. This whole representation is used as the torso, which inputs the current graph and outputs a representation, which is consumed by the policy and value heads.

The current graph is given input as an  $n \times n$  adjacency matrix. Since we use a single network for multiple sizes, we condition the torso and the policy head on the graph size  $n$  by concatenating each input with a matrix of 1s on the principal  $n \times n$  submatrix and 0s everywhere else (concatenate along the channel dimension). This lets us use a shared set of parameters for multiple graph sizes without a separate network for each size.

A good model architecture should not only be expressive but also have fast inference, so that acting would be fast enough to quickly generate lots of data for the learner to optimize the model. The downside of pairformer is its  $O(n^3)$  runtime, while resnet’s runtime is  $O(n^2)$ . Hence there exists a trade-off, and we experimentally found that combining a small pairformer torso with a larger resnet policy head provides the best balance. Using a resnet for the torso performs much worse, indicating that the expressiveness which pairformer brings to the torso is indispensable (for details, see Appendix E). For the value head, we used a multi-layer perceptron over the representation provided by the pairformer.

Another important detail is that, although the environment supports only  $\binom{n}{2}$  many actions, the last layer of our policy network has twice that many logits: for each pair  $(i, j)$  of nodes, there is one logit for adding that edge and another logit for removing that edge. This means half of the logits correspond to dummy actions. We mask these dummy actions so a valid probability distribution is induced on the set of  $\binom{n}{2}$  actions.

#### D.4 Choosing the episodes’ starting state

Starting episodes from the empty graph leads to a difficult credit assignment problem for the RL agent, as the horizon should be long. Instead, we start from a high-scoring graph of size  $n - k$  to build the target graph of size  $n$ . We observe that this choice of initial graph is critical for the performance of AlphaZero. It leads to a shorter horizon and more effective credit assignment in each episode.

#### D.5 Distributed implementation and joint learning across multiple sizes

We use a distributed implementation of AlphaZero with multiple processing units: in each run, there are multiple actors, one replay buffer, and one learner. Each actor has a copy of the networks (supplied by the learner) and generates episodes, which are inserted into the replay buffer. The learner repeatedly samples an episode from the replay buffer to update the policy and value networks. The policy network is trained using the cross-entropy loss, where the ground truth label is assumed to be the decision taken at the root of the MCTS tree. The value network is trained using regression on the future return (sum of the future rewards) at each state of an episode.

For efficiency and transfer-learning across multiple sizes, we jointly train a single network for multiple sizes: each run of AlphaZero is provided with a list of target sizes, and each actor samples a target size uniformly at random from this list. The network input is modified in this case by padding it by 0s to turn it into a  $target \times target$  matrix, while appending another plane to the observation, each entry of whose principal  $size \times size$  submatrix is 1 and the rest are 0. This helps to run experiments for multiple sizes jointly and ensures transfer-learning across different sizes.

#### D.6 Other implementation details

For each size  $n$ , we started the episodes with one of the high-scoring graphs found by incremental tabu search at size  $n - k$ , where  $k$  is chosen randomly between 1 and 4. (We did this for technical



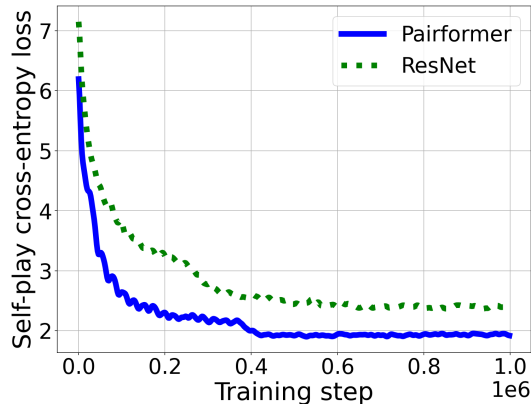


Figure 3: The policy cross-entropy loss of pairformer and resnet during online training of AlphaZero (Gaussian smoothing with  $\sigma = 2$  is applied) on joint training for graph sizes [80, 100] with curriculum. Pairformer minimizes the loss faster as it captures invariances and other structure in the graph.

convenience, but we believe similar results are achievable if we sample from graphs of smaller sizes generated by previous runs of AlphaZero.)

A key hyperparameter is the episode length (the horizon). An overly long episode length is not only wasteful but also hinders learning, as the agent may find an optimal graph in the middle of an episode but still has to flip edges to reach the end of the episode. On the other hand, an overly short episode length would hinder the exploration as the agent is limited to the vicinity of the initial graph. We ran AlphaZero without curriculum for sizes 5 to 100, split these sizes in five nearly-equal buckets of 5–20, 21–40, 41–60, 61–80 and 81–100, and set horizons to 80, 160, 240, 320, and 434, respectively. With curriculum, since we start from a good graph of smaller size, we experimented with horizon lengths of 30, 50 and 100 but found the results don’t change much beyond 30. The initial graph is sampled from all possible graphs of smaller size from [5, n-1].

## E Comparing representations in AlphaZero

We compared our novel pairformer representation with resnet[20], which has been extensively used in literature, especially in environments where the observation is a matrix or an image. Since a graph can be naturally expressed as an adjacency matrix, we compare the resnet architecture, which is oblivious to the graph structure, with the pairformer architecture. We use resnet with 10 layers and 256 output channels. Figure 3 shows the policy’s cross-entropy loss during training, and Figure 4 shows the average episode return. For this experiment, we focus on joint training for graph sizes [80, 100] with curriculum. We observe that pairformer performs better on both metrics. Nevertheless, the final scores obtained by resnet and pairformer are equal except on sizes 80, 86 and 93, where pairformer leads by one score point. It should be noted that the above experiments are for parameters sizes (number of layers, attention heads) beyond which we were not able to see any performance improvement for both the representations (We explicitly didn’t compare resnet and pairformer with exactly same number of parameters).

## F Incremental tabu search

Tabu search is a well-known iterative local search method [19]: given an objective function and a neighborhood structure over a set of states, it repeatedly moves from the current state to the neighboring state with the highest objective value, until some stopping condition is met. To avoid getting stuck at local minima, tabu search bans revisiting recently-visited states—hence the name “tabu” search.

In our case, the states are the graphs of a given size, the graphs obtained by flipping a single edge are the neighbors of the current graph, and the objective function is  $s(G) = e(G) - \Delta(G) - \square(G)$ . Our tabu search algorithm—see Algorithm 1—slightly differs from the typical definition; instead

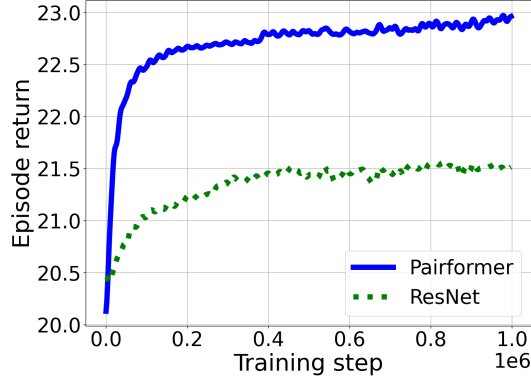


Figure 4: Average episode return of pairformer and resnet during online training of AlphaZero using the edge-flipping environment (Gaussian smoothing with  $\sigma = 2$  is applied) on joint training for graph sizes [80, 100] with curriculum. The average is taken over three seeds.

of banning visiting *states* that were recently visited, we ban playing the recently-played *actions*. Namely, we ban re-flipping edges that were flipped recently. This idea, inspired by Parczyk et al. [32], results in a slightly faster algorithm than the usual tabu search. Note that the algorithm needs an initial graph  $G_0$ —we will describe later how it’s chosen—and has a single hyperparameter: the history size, denoted by  $h$  in Algorithm 1, which determines the number of iterations flipping an edge is banned once it is flipped. Recall that  $\oplus$  denotes flipping an edge.

---

**Algorithm 1** Our version of tabu search

---

**Require:**  $G_0$  is an  $n$ -node graph with nodes indexed from 1 to  $n$ , and  $0 \leq h < \binom{n}{2}$

**Ensure:**  $BestGraph$  is the highest-scoring graph found during search

- 1:  $Tabu \leftarrow$  a first-in-first-out queue of fixed size  $h$
  - 2:  $Actions \leftarrow \{(i, j) : 1 \leq i < j \leq n\}$
  - 3:  $BestGraph \leftarrow G_0$
  - 4: **for**  $i \leftarrow 1, 2, \dots, iterations$  **do**
  - 5:    $ValidActions \leftarrow Actions \setminus Tabu$
  - 6:    $BestActions \leftarrow \arg \max_e \{s(G_{i-1} \oplus e) : e \in ValidActions\}$
  - 7:    $Action \leftarrow$  random action chosen from  $BestActions$
  - 8:    $G_i \leftarrow G_{i-1} \oplus Action$
  - 9:   Insert  $Action$  into  $Tabu$
  - 10:   **if**  $s(G_i) > s(BestGraph)$  **then**
  - 11:      $BestGraph \leftarrow G_i$
  - 12:   **end if**
  - 13: **end for**
- 

The *incremental* tabu search algorithm is inspired by the idea described in Section 3: we let the tabu search at each size start its search from one of the best graphs found at smaller sizes. Say we want to find lower bounds for  $f(n)$  for some range  $n \in \{a, \dots, b\}$ . Incremental tabu search is a distributed algorithm with  $b - a + 1$  parallel workers (processing units), indexed from  $a$  to  $b$ , where the worker with index  $n$  searches for graphs of size  $n$ . The workers need a common memory to share the graphs they have found: suppose that  $BestGraphs[n]$ , for  $n \in \{a, \dots, b\}$ , is a set of graphs that all workers have access to. We initialize it to contain just the empty graph of size  $n$ . The algorithm for the size- $n$  worker appears in Algorithm 2.

For the experiments, for tabu search, we experimented with history sizes 1, 5, 10, and 20; size 5 worked best. For each size, we ran 32 parallel copies of tabu search for seven days, restarting every 1000 iterations and merging the results. For incremental tabu search, we initialized  $BestGraphs[n]$  (see Algorithm 2) to contain the set of graphs published by McKay [30] (for  $n = 1, 2, \dots, 64$ ), set the history size to 5, and the  $K$  in incremental tabu search to 4—it is important this is greater than 1.

---

**Algorithm 2** Incremental tabu search (worker for size  $n$ )

---

**Require:**  $0 \leq K$  and  $a \leq n \leq b$  and  $BestGraphs[n]$  is initialized to the empty set

**Ensure:**  $BestGraphs[n]$  contains the list of highest-score graphs found during the search

```
1: while True do
2:   Sample  $k$  randomly from  $\{1, \dots, K\}$ 
3:   Sample  $G_0$  randomly from  $BestGraphs[n - k]$ 
4:   Add  $k$  isolated nodes to  $G_0$ 
5:   Run tabu search starting from  $G_0$ 
6:    $BestFoundGraph \leftarrow$  best graph found by tabu search
7:   Choose  $ExistingGraph$  arbitrarily from  $BestGraphs[n]$ 
8:   if  $s(BestFoundGraph) > s(ExistingGraph)$  then
9:      $BestGraphs[n] \leftarrow \{BestFoundGraph\}$ 
10:  else
11:    Add  $BestFoundGraph$  to  $BestGraphs[n]$ 
12:  end if
13: end while
```

---

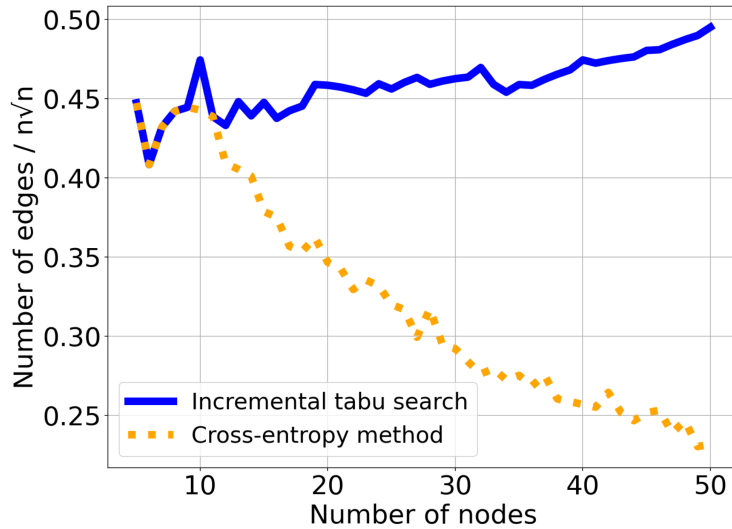


Figure 5: Incremental tabu search performs much better than the cross-entropy method.

## G Comparing the cross-entropy method with incremental tabu search

We compare incremental tabu search to the cross-entropy method [44] in Figure 5. For the cross entropy, we performed a hyperparameter sweep over the values in Table 3 for each graph size.

Table 3: The hyperparameter sweep for the cross-entropy method [44]. “Examples per iteration” refers to the number of proposed constructions to sample on each iteration before sampling the top  $k$  percent and training the neural network on those highest scoring examples.

Learning rate	Examples per iteration	Top $k$
1e-4, 1e-5	100, 1000	5, 8, 10