Learning Lower Bounds for Graph Exploration With Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

We explore the usage of reinforcement learning for theoretical computer science. Reinforcement learning has shown to find solutions in challenging domains such as Chess or Go. Theoretical problems, such as finding the worst possible input for an algorithm come with even more vast, combinatorial search spaces. In this paper, we look at the example of online graph exploration. Here we want to find graphs that yield a high competitive ratio for a greedy explorer. The search space consists of having every edge being either present or absent. Given there are quadratically many possible edges in a graph and each subset of edges is a possible solution, this yields unfeasibly large search spaces even for few nodes. We show experimentally how clever constraints can keep such search spaces manageable. As a result, we can learn graphs that resemble those known from literature and even improve them to yield higher competitive ratios.

1 Introduction

We have seen reinforcement learning achieve some remarkable results in recent years. Some of the remarkable results, in our opinion, show the ability to navigate search spaces for games such as Chess or Go [Silver et al., 2017], achieving superhuman performance. While these games have perfect information, the models evaluate positions deeply, which requires navigating a combinatorially exploding search space. At the same time, we have seen that machine learning finds solutions to classical difficult combinatorial problems such as the traveling salesman problem [Khalil et al., 2017] or the satisfiability problem [Selsam et al., 2019]. These papers show that machine learning is promising in finding solutions, even when we know the problems are difficult ($\mathcal{NP}$-hard) and come with large combinatorial spaces. In this paper, we turn this setting around. We look at a problem, that is not yet fully understood (graph exploration). For graph exploration, we want to more closely understand the limits of a particular solution (greedy exploration). To understand this better, let us first give a brief introduction to (online) graph exploration.

Online Graph Exploration. Online graph exploration is closely related to the traveling salesman problem (TSP). Both problems have an explorer that wants to visit all nodes in the graph while minimizing the edge traversals. In TSP, we are given the complete graph as input. On the other hand, graph exploration is an online algorithm. That means the input graph is gradually revealed as we walk around. At every point, the explorer only received information about the explored nodes and which nodes are adjacent to them (see Figure 1 for an example). The relevant metric for comparing the two problems is the *competitive ratio*. This ratio measures the ratio of the best possible online graph exploration algorithm versus the best possible offline graph exploration algorithm (i.e., a TSP solution). That is, how much more difficult the problem becomes by not having full information about the input from the very beginning.

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

(a) Information in travelling salesman

(b) Information in graph exploration

Figure 1: Comparison of information for travelling salesman (left) and graph exploration (right). Dark nodes are explored, bright nodes part of the input. Non-colored nodes and dashed edges are not part of the input.

From the perspective of competitive ratio, graph exploration is not yet well understood. On the one hand, we know that the competitive ratio is at least $\frac{10}{3}$ [Birx et al., 2020] which is a very recent improvement over the old lower bound of 2.5 [Dobrev et al., 2012]. This bound means that no matter the exploration algorithm, there exists a graph where the offline computed TSP tour will be shorter by a factor of at least $\frac{10}{3}$. The theoretical gap here is that we do not know how "correct" this factor is or even if this factor is really independent of the number of nodes. For a particular approach to solve graph exploration, we know more: the greedy explorer is an intuitive, reasonable heuristic that always goes to the nearest unexplored node. For greedy we know, that its competitive ratio is indeed dependent on the graph size; the greedy explorer has a competitive ratio of $\Theta(\log n)$ [Hurkens and Woeginger, 2004].

Reinforcement Learning for Online Graph Exploration. In this paper, we take a closer look at this asymptotic bound, and we want to better understand this bound in terms of constant factors hidden inside the asymptotic notation. In this paper, we focus our analysis on unweighted undirected graphs. Hurkens and Woeginger [2004] show a construction yielding a competitive ratio of $\frac{1}{4}(3 + \log n)$. We explore if we can improve this lower bound using reinforcement learning. Note that for a graph with n nodes, there are $\frac{n \cdot (n+1)}{2}$ many possible edges, each of which can be independently present or not. Thus, the search space is in the order of $2^{\Theta(n^2)}$ and explodes even for small values of n. To add to this challenge, we are only interested in the few bad solutions which constitute a small fraction of the state space. Thus, we present heuristics to keep this search space manageable. Indeed, experiments show that we can discover structures such as those in Hurkens and Woeginger [2004] and modify them to reach larger competitive ratios.

2 Related Work

Reinforcement Learning for Combinatorial Problems. Previous work has used machine learning, and in particular reinforcement learning, to provide good solutions for several combinatorial and NP-hard problems [Khalil et al., 2017, Vinyals et al., 2015, Bello et al., 2017, Selsam et al., 2019, Amizadeh et al., 2019]. In these works, the authors give a well-understood problem class and use machine learning to solve instances of these problems. The other way around, Sato et al. [2019] want to learn a generative model that constructs difficult problem instances of a difficult problem class. Our work is more in the second spirit, we learn instances that are expensive for a given algorithm (the greedy explorer) to solve.

Algorithm Inference with Machine Learning. In the related field of algorithm inference, learning-based methods infer algorithmic structures from input-output examples and execution traces. The learning is enhanced with new architectures [Kaiser and Sutskever, 2016, Freivalds and Liepins, 2017, Neelakantan et al., 2016, Velickovic et al., 2020], differentiable storage components [Graves et al., 2014, 2016, Grefenstette et al., 2015, Zaremba and Sutskever, 2016], composition of smaller sub-programms [Reed and de Freitas, 2016, Zaremba et al., 2016], recursion Cai et al. [2017], or logic reasoning [Evans and Grefenstette, 2018, Dong et al., 2019]. The case of this paper instead is to learn more about the limitations of a known, given algorithm.

3 The Reinforcement Learning Model

We learn a construction agent (constructer) to create a graph with a maximal competitive ratio against an exploring agent (explorer). Principally, the constructer plays the explorer in rounds in a problem-setter versus problem-solver framework. The constructer is given the current position (a node) of the explorer and decides how to connect this node to the rest of the graph. Then, the explorer decides which node to visit next. In our scenario of the greedy explorer, it computes the distances to all nodes that are adjacent to at least one explored node and moves to the closest one. In the case of ties — that might happen since we look at unweighted graphs — the explorer tiebreaks to the node with a lower node ID. After moving to this node, the constructer takes another turn to connect this node, ... until all nodes have been explored.

State Space. We design the state space to provide "full" observability, where full refers to all the available information of the graph reinforcement learning framework. First, the constructer receives the ID of the node being currently explored. Nodes have ascending IDs from 0 to $n-1$; the current node is given in a one-hot vector. Second, the constructer receives a bit vector of length n indicating whether the node was already explored or not. Finally, the constructer receives the edges of the current version of the graph in the form of an adjacency matrix. In the scope of this paper, we follow the model of unweighted, undirected graphs. Because of symmetry, we provide only the upper half of the adjacency matrix which contains bit values if an edge is present or not. An extension to directed graphs (we provide the full adjacency matrix) as well as weighted graphs (the adjacency matrix contains numbers instead of bits) could be explored in future work.

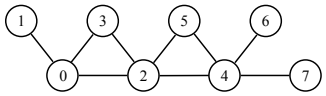
Action Space. In every step, the constructer decides how to connect the current node c of the explorer to the rest of the graph. We feed the state vector into a neural network, whose outputs are n decision variables, where the i-th variable encodes if there should be an edge from c to node i. We mask self edges ($c = i$) and ignore decisions on the pair c, i if we made a decision on i, c before. This is because we focus on an undirected, that is a symmetric, graph. In the case of an directed graph, these decisions would be independent.

Managing the state space dimensionality. Even with uniform weights and undirected edges, there are $\frac{n\cdot(n-1)}{2}$ possible edges. Each subset of those edges is a valid solution, giving the constructer a total of $2^{\frac{n\cdot(n-1)}{2}}$ possible graphs to search. Exploring this state space becomes very quickly infeasible, even for moderately small numbers of nodes. Thus, we employ heuristic constraints on the state space. Let us consider the initial situation, where the constructer connects node 0 to other nodes. Principally, the exact nodes to connect to are less important than the number of nodes. All different combinations of a certain number of edges lead to isomorphic graphs that largely behave the same (apart maybe from some cases of tiebreaking). However, the constructer is oblivious to this isomorphism and has to learn the properties of every graph individually. For adding k edges, there are $\binom{n\cdot(n-1)/2}{k}$ many possibilities. We propose to choose one canonical representative for this set of graphs that we use consistently instead. This collapses the all permutations to just one, giving an *exponential* speedup in the search space — *per round*.

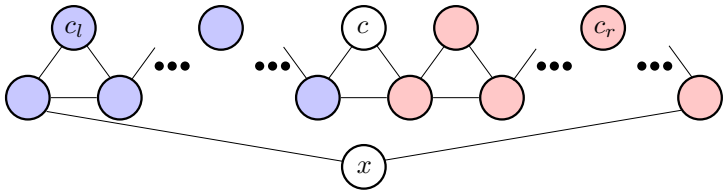
Concretely, let V be the set of nodes in the graph. We can partition V into 3 sets: the set E of all nodes that have been explored, the set B of all nodes adjacent to nodes in E, and the set of remaining nodes U. Nodes in U are those that the explorer does not know about, yet. This is the property we exploit. We add a new output to the action space that decides how many edges in U the newly explored node is connected to. Then we pick this number of nodes in U in a canonical way — in this paper we take the lowest numbered nodes from U. The decision variables for nodes in U are ignored.

4 Experimental Investigation

We investigate the efficacy of this state space model, action space model, and search space heuristic by experimental tests. In particular, we want to investigate if our constructer can find graphs that match or surpass the competitive ratio of known theoretical constructions. We experiment with different graphs sizes $n = 2^i$ for different i. We follow the known constructions, such as Hurkens and Woeginger [2004] that use graphs with where the number of nodes is a power of two. We learn the neural networks and outputs with proximal policy optimization [Schulman et al., 2017]. One



(a) Example triangle structure found by the constructer.



(b) Recursive triangle structure. Two smaller structures (left in blue, right in red) are merged through node c. Node x connects the left and right ends.

episode is a full exploration cycle when the explorer visited every node for which we give a reward of the achieved competitive ratio minus 1 (ratio 1 means the online algorithm was just as good as the offline algorithm). In total, we train constructers for 500.000 decisions, which gives a differing amount of episodes depending on the graph size.

We found that graph construction using the vanilla architecture without the heuristic search space constraints does not allow us to find graphs with a high competitive ratio, apart from very small graphs with $n = 8$ nodes. On the other hand with the heuristic, we find good graphs for larger sizes using the heuristic of having canonical representatives for isomorphic graphs. In general, the graphs we find there follow a triangle structure as in Figure 2a. This structure resembles the construction from Hurkens and Woeginger [2004].

Triangle Improvements In the second line of experiments, we tried if we can improve on the construction of Hurkens and Woeginger [2004]. They employ a recursive structure. The base graph is a triangle with three nodes. Upper levels are constructed by combining two lower-level structures and merging them to a long chain of triangles, adding a new center node. They also add a node x to connect the leftmost and rightmost nodes. Refer to Figure 2b for a visualization.

In this set of experiments we run the constructer but mask the edges from Hurkens and Woeginger [2004] to 1, such that at least these edges are always present. This structure imposes a strong bias on the constructions for the constructer, so we do not employ our heuristic as well. We find that our constructer learns to improve on the provided structure systematically.

The idea lies in misleading the greedy explorer to finish the exploration. Consider the situation when the explorer already explored both recursive structure, thus is at node c_r. Now there are three steps left to do: explore c and x in any order and return to the starting node l. In the construction of Hurkens and Woeginger [2004], the explorer will first explore c, then x, then return to l. The reinforcement learning agent learns that exploring x first and then c is better because now it costs $\frac{n}{4}$ to move back to l in the end, instead of 1 if c was explored first. In the original construction, the nodes x and c are equidistant, so the agent learns to create a "shortcut" in the right half. This shortcut connects two nodes that are two hops away directly. Now x is closer to c_r, and greedy explores there first. On the other hand, this shortcut might also compromise earlier exploration; however, we found that shortcuts that provide just this intended effect exist. With this approach we are able to improve the known competitive ratio from $\frac{1}{4}(3 + \log n)$ to $\frac{1}{4}(4 + \log n) - \frac{3}{n}$.

5 Conclusions and Outlook

We see many interesting avenues to continue this line of research. On the one hand, much remains unexplored in terms of online graph exploration. Future work could look at training constructers against other exploration algorithms than greedy. For example, blocking [Kalyanasundaram and Pruhs, 1994, Megow et al., 2012] is a proposed algorithm that is supposed to perform better than the greedy explorer. One more alternative would be to realize the explorer through another reinforcement learning agent and learn both constructers and explorers with self-play. Last, we saw that the constructer had difficulties catching recursive structures, this might be a promising area to explore further modeling approaches that come with recursive inductive biases.

References

S. Amizadeh, S. Matusevych, and M. Weimer. Learning to solve circuit-sat: An unsupervised differentiable approach. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. Neural combinatorial optimization with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=Bk9mxlSFx`.

A. Birx, Y. Disser, A. V. Hopp, and C. Karousatou. Improved lower bound for competitive graph exploration. In *Department of Mathematics, TU Darmstadt, Germany*, 2020.

J. Cai, R. Shin, and D. Song. Making neural programming architectures generalize via recursion. In *5th International Conference on Learning Representations (ICLR), Toulon, France*, Apr. 2017.

S. Dobrev, R. Královič, and E. Markou. Online graph exploration with advice. In *Aceto L., Henzinger M., Sgall J. (eds) Automata, Languages and Programming. ICALP 2011. Lecture Notes in Computer Science, vol 6756. Springer, Berlin, Heidelberg*, 2012.

H. Dong, J. Mao, T. Lin, C. Wang, L. Li, and D. Zhou. Neural logic machines. In *7th International Conference on Learning Representations (ICLR), New Orleans, USA*, May 2019.

R. Evans and E. Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 2018.

K. Freivalds and R. Liepins. Improving the neural gpu architecture for algorithm learning. *arXiv preprint arXiv:1702.08727*, Feb. 2017.

A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, Dec. 2014.

A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 2016.

E. Grefenstette, K. M. Hermann, M. Suleyman, and P. Blunsom. Learning to transduce with unbounded memory. In *Advances in neural information processing systems*, 2015.

C. A. J. Hurkens and G. J. Woeginger. On the nearest neighbor rule for the traveling salesman problem. In *Operations Research Letters Volume 32, Issue 1*, July 2004.

L. Kaiser and I. Sutskever. Neural gpus learn algorithms. In *4th International Conference on Learning Representations (ICLR), San Juan, Puerto Rico*, Apr. 2016.

B. Kalyanasundaram and K. R. Pruhs. Constructing competitive tours from local information. In *Theoretical Computer Science Volume 130, Issue 1*, Aug. 1994.

E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358, 2017.

N. Megow, K. Mehlhorn, , and P. Schweitzer. Online graph exploration: New results on old and new algorithms. In *Even G., Halldórsson M.M. (eds) Structural Information and Communication Complexity. SIROCCO 2012. Lecture Notes in Computer Science, vol 7355. Springer, Berlin, Heidelberg*, 2012.

A. Neelakantan, Q. V. Le, and I. Sutskever. Neural programmer: Inducing latent programs with gradient descent. In Y. Bengio and Y. LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL `http://arxiv.org/abs/1511.04834`.

S. E. Reed and N. de Freitas. Neural programmer-interpreters. In *4th International Conference on Learning Representations (ICLR), San Juan, Puerto Rico*, Apr. 2016.

R. Sato, M. Yamada, and H. Kashima. Learning to sample hard instances for graph algorithms. In *Asian Conference on Machine Learning*, pages 503–518, 2019.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill. Learning a SAT solver from single-bit supervision. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL `https://openreview.net/forum?id=HJMC_iA5tm`.

D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

P. Velickovic, R. Ying, M. Padovano, R. Hadsell, and C. Blundell. Neural execution of graph algorithms. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=SkgKO0EtvS`.

O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In *Advances in neural information processing systems*, pages 2692–2700, 2015.

W. Zaremba and I. Sutskever. Reinforcement learning neural turing machines-revised. *arXiv preprint arXiv:1505.00521*, Jan. 2016.

W. Zaremba, T. Mikolov, A. Joulin, and R. Fergus. Learning simple algorithms from examples. In *33rd International Conference on Machine Learning (ICML), New York City, USA*, June 2016.