Annals of Mathematics and Artificial Intelligence
https://doi.org/10.1007/s10472-022-09813-2

Learning from obstructions: An effective deep learning approach for minimum vertex cover

Faisal N. Abu-Khzam[1] · Mohamed M. Abd El-Wahab[2] · Moussa Haidous[1] · Noureldin Yosri[3]

Abstract
Computational intractability has for decades motivated the development of a plethora of methodologies that mainly aim at a quality-time trade-off. The use of Machine Learning has finally emerged as one of the possible tools to obtain approximate solutions to $\mathcal{NP}$-hard optimization problems. Recently, Dai et al. introduced a method for computing such approximate solutions for instances of the Vertex Cover problem. In this paper we consider the effectiveness of selecting a proper training strategy by considering special problem instances called *obstructions* that we believe carry some intrinsic properties of the problem. Capitalizing on the recent work of Dai et al. on Vertex Cover, and using the same case study as well as 19 other problem instances, we show the utility of using obstructions for training neural networks. Experiments show that training with obstructions results in a surprisingly huge reduction in number of iterations needed for convergence, thus gaining a substantial reduction in the time needed for training the model.

Keywords Deep learning · Vertex cover · Graph minor Theorem

Mathematics Subject Classification (2010) 68T01

✉ Faisal N. Abu-Khzam
faisal.abukhzam@lau.edu.lb

Mohamed M. Abd El-Wahab
mohamedmahmoudabdelwahabmohamed.mahmoud@cdu.edu.au

Moussa Haidous
moussa.haidous@lau.edu.lb

[1] Lebanese American University, Chouran, Beirut 1102 2801, Lebanon

[2] School of Engineering, Information Technology & Environment, Charles Darwin University, Darwin, Northern Territory, Australia

[3] College of Computing and Information Technology, Arab Academy of Science and Technology, Alexandria, Egypt

1 Introduction

Let G be a finite undirected graph with possible loops and multiple edges. A graph H is said to be a minor of G, or $H \leq_m G$, if a graph isomorphic to H can be obtained from a subgraph of G by contracting zero or more edges. In this context, an edge uv is contracted simply by replacing u and v with a new vertex w whose neighborhood is the union of the neighborhoods of u and v. As such, $\leq_m$ defines a partial ordering on the set of finite graphs known as the minor order.

Let F be a family of finite undirected graphs that is closed under the minor order. The obstruction set of F is defined as the set of graphs in the complement of F that are minimal in the minor order. That is X is an obstruction of F if $X \notin F$ and every minor of X is in F. A typical classical example of a minor-closed family is the class of planar graphs. In this case, and according to Wagner's theorem [32], the set of obstructions consists of only two graphs.

Thanks to the celebrated Graph Minor Theorem, which states that finite graphs are well quasi ordered by the minor order relation [28], we know the obstruction set of a minor-closed family is finite. This implicitly leads to polynomial-time membership tests for such families, though non-constructively according to the work of Fellows and Langston [16], which paved the way for the emergence of *parameterized complexity* theory [15]. The main idea being that a decision problem whose family of YES-instances is minor-closed would be efficiently solvable in polynomial time when some input parameter is fixed.

A well studied notable example is the classical Vertex Cover problem, which is used as a case study in this paper. Unfortunately, there is no general algorithm for constructing sets of vertex-cover obstructions [13]. In fact, Fellows and Langston showed that computing obstruction sets for minor-closed graph families is recursively unsolvable in general [16, 17]. Despite this fact, many attempts have been made to compute problem-specific obstructions for several minor-closed families [3, 7, 9, 13, 23]. In this paper we address the optimization version of the Vertex Cover problem and show that computing a potentially large subset of such obstructions can be used to improve vertex cover approximation via a proper use of deep learning.

Using deep learning to solve graph theoretic problems is considered hard in general, even if approximate solutions are sought. This can be attributed to the difficulty of representing graphs in a grid-like structure, and deal with various forms of graphs: weighted/unweighted, directed/undirected, etc. Moreover, the nature of a required solution varies from one problem to another (vertex/edge classification, graph classification, etc.) To tackle these challenges, different neural network architectures have been proposed in the literature, including Graph Neural Networks (GNNs) [18, 20, 30], Graph Convolution Networks (GCNs) [5], and Graph Autoencoders (GAEs) [24].

An in depth discussion of graph problems to which deep learning can be applied was recently presented by Zhang et al. in [33]. To tackle combinatorial optimization problems on graphs, Vinyals et al. introduced the Pointer Net (Ptr-Net) architecture [31], while Bello et al. proposed a reinforement learning approach to combinatorial problems [4].

In this paper, we build on the work of Dai et al. in [10] and [11], namely using the Structure2Vec model and the reinforcement learning formulation of Bello et al. in [4]. The paper is structured as follows. Section 2 provides some preliminaries and background material including known Vertex Cover approximation algorithms; Section 3 describes the deep learning approach; in Section 4 we describe how sets of small connected Vertex Cover obstructions are constructed; In Section 5 we present our experiments, and we conclude with a summary and future directions in Section 6.

2 Preliminaries

Throughout the paper we use common graph theoretic terminology. For a graph G, we use $V(G)$ and $E(G)$ to denote the sets of vertices and edges of G, respectively. For a vertex v of G, we denote by $N(v)$ the neighborhood of v, i.e., the set of vertices adjacent to v in G, and by $E(v)$ the set of all edges that are incident on v.

A vertex cover of a graph G is a subset C of the vertex set of G whose deletion results in an edge-less graph. In other words, every edge of G has at least one endpoint in C. The corresponding optimization problem, which seeks a minimum-cardinality vertex cover in a given graph, is one of the most studied classical $\mathcal{NP}$-hard problems [21].

An effective greedy approach for Vertex Cover consists of successively selecting a vertex of maximum degree and placing it in the solution, until the graph is edge-less. A pseudocode is given below.

```
$C \leftarrow \phi$
while there is at least one uncovered edge do
    $u \leftarrow argmax_{v \notin C} \| \{vt | t \notin C\} \|$
    $C \leftarrow C \cup \{u\}$
end while
Return $C$
```

The above method does not guarantee a constant-factor approximation, but it is known to perform well in practice. In fact the problem is known to be $\mathcal{APX}$-complete with a best-known factor-two polynomial-time approxiamtion algorithm, attributed to F. Gavril in [19], but also known to be independently discovered by M. Yannakakis. A pseudocode of this latter approximation algorithm follows.

```
$C \leftarrow \phi$
while there is at least one uncovered edge do
    $uv \leftarrow$ any uncovered edges
    $C \leftarrow C \cup \{u, v\}$
end while
Return $C$
```

This simple factor-two approximation algorithm has been shown to be best-possible modulo the unique games conjecture [22]. The same approximation ratio can also be obtained via the problem's ILP formulation [27], which was also shown to be equivalent to computing a special structure known as "crown" in the input graph [2, 8]. However, the above approximation algorithm remains simpler and much faster.

In the next section, we present a reinforced machine learning approach that can compete with the above approximation algorithms.

For any (fixed) integer k, the family F_k of graphs of minimum vertex cover bounded above by k is minor-closed. This is simply due to the fact that neither edge contraction nor vertex/edge deletion can increase the minimum vertex cover size. Therefore we know the set of obstructions for F_k is finite. We shall denote this set by $\mathcal{O}b(k)$ in what follows. We also use $vc(G)$ to refer to a minimum-size vertex cover of G.

We should note that many combinatorial graph problems satisfy the same closure property (under taking minors) when fixing some parameter that is often the solution size. Notable examples include Treewidth, Pathwidth and Feedback Vertex Set. We shall refer to such problems as being *minor-closed* in the rest of this paper.

We denote by C_n the cycle on n vertices, and by K_n the complete graph on n vertices. Observe that a graph G that has C_{2k+1} as minor must have a minimum vertex cover of size strictly greater than k. The same applies if $K_{k+2} \leq_m G$. Thus $\mathcal{O}b(k)$ contains C_{2k+1} and K_{k+2}.

Unfortunately, it was shown in [6] and [13] that the size of the obstruction set for k-Vertex Cover grows exponentially with k. In general, there is no constructive approach to generate all the connected obstructions of k-Vertex Cover from the obstructions for $(k-1)$-Vertex Cover [13]. However, a large percentage of such obstructions can be obtained, especially when k is small enough, as we shall see in the sequel.

3 Deep learning for minimum vertex cover approximation

In [11], Dai et al. approximated solutions to a number of combinatorial optimization problems including Minimum Vertex Cover. The main setup formulates the problem as a reinforcement learning problem as in [4]. It is then applied to graphs generated using the Erdos-Renyi model, in addition to the real-world dataset Meme-Tracker.[1]

We further put forward that it is true when training a network on a specific family of graphs such as $Ob(k)$, which share the same statistics ($MVC = k + 1$), that the network should be able to learn those statistics and to maximize a function mapping the given choices for a move (e.g. selecting the next vertex to add to the vertex cover) to a cost representing the true cost/gain of adding that vertex. With this in mind, we propose that the set of Vertex Cover obstructions is a suitable family of graphs for the minimum Vertex Cover problem.

We shall use the same network architecture as [11], dubbed S2V-DQN. Initially, the graphs are embedded in the latent space using Structure2Vec [10] and then used to train a deep Q-learning model.

3.1 Structure2Vec

The Structure2Vec method computes a p-dimensional feature embedding μ_v for each vertex $v \in V$. It does so by aggregating node-specific tags x_v according to the graph's topology, and hence preserving its structure. It first initializes $\mu_v^0 = 0$ and then updates the embedding at each iteration as such:

$$\mu_v^{t+1} \leftarrow F\big(x_v, \{\mu_u^t\}_{u \in N(v)}, \{w(u,v)\}_{u \in N(v)}, \Theta\big)$$

where $N(v)$ is the set of neighbors of v in the graph, $w(u, v)$ is the weight of the edge uv, Θ is the set of parameters to be learned, and F is a non-linear mapping such as a neural network in our setup.

It could then be seen that the node-specific tags x_v are propagated throughout the graph using F. Hence, after T iterations of F at each node, every node embedding μ_v^T carries information about its T-hop neighborhood hence maintaining the graph's structure.

[1] http://www.memetracker.org/

3.2 Reinforcement learning

To construct the solution S for the minimum Vertex Cover problem, a greedy algorithm would choose a node v to be added to the solution set according to some evaluation function Q. Typically, Q is hand-crafted and could be as trivial as choosing the node with the highest degree. Instead, utilizing deep learning, an evaluation function $\hat{Q}(S, v, \Theta)$ could be learned, as is done in [10].

The authors of [11] show the evaluation function $\hat{Q}$ lends itself to a reinforcement learning framework. Hence, the problem can be formulated as follows:

1. *States*: A sequence S of nodes embedded in p-dimensional space such that $S = \Sigma_{v \in V} \mu_v$.
2. *Transition*: Tagging selected node $v \in G$ with 1 such that $x_v = 1$
3. *Action*: Add node $v \in G$ that is not in the partial solution $\hat{S}$ to $\hat{S}$.
4. *Reward*: $r(S, v) = -1$
5. *Termination*: all edges are covered.

A combination of n-step Q-learning and *fitted* Q-iteration is used to carry out the training. Specifically, an agent A attempts to build the vertex cover by choosing a node v and adding it to the solution set. Node v is either chosen randomly (to allow agent exploration) or chosen according to policy π depending on some ϵ that dictates the likelihood of choosing node v randomly as opposed to according to π.

The reward $r(S, v) = -1$ is chosen so that the agent is discouraged from adding nodes and hence minimizing the size of the vertex cover (maximizing the reward). We note that n-step Q-learning helps with the issue of delayed rewards where the final objective value of solving the Minimum Vertex Cover instance is revealed after multiple node additions. Opposed to 1-step Q-learning's target value $y = r(S_t, v_t) + \gamma max_{v'} Q(S_{t+1}, v', \Theta)$, n-step Q-learning updates the network's parameters every n-steps with a target value of $y = \Sigma_{i=0}^{n-1} r(S_{t+i}, v_{t+i}) + \gamma max_{v'} Q(S_{t+n}, v', \Theta)$.

Furthermore, fitted Q-iteration updates the Q-function with a random batch of samples by using *experience replay* instead of updating the Q-function over just one sample, which lessens the effect of the correlation between samples that are one time step apart.

After the agent takes an action (adding a vertex to the set), the tuple $(S_t, a_t, R_{t,t+n}, S_{t+n})$ is added to a set E with $R_{t,t+n} = \Sigma_{i=0}^{n-1} r(S_{t+i}, a_{t+i)}$. $R_{t,t+n}$ is what permits the utility of n-step Q-learning.

Finally, the tuple describes the current solution set (state), which node v was added to the set (action), the sum of rewards up until $t + n$, and the solution set reached after n-iterations.

4 Vertex cover obstructions

Selecting a proper set of graphs as problem instances for training is our main objective in this paper. Random samples are often used, and they have been used in [11]. It is obvious that training on random samples requires a large number of epochs. Two questions pose themselves naturally in this context:

First, what properties would a problem instance have in order to make it "informative" from a training perspective? Secondly, how to find a small set of representative instances that can potentially guide the learning and make it more efficient?

Although hard to answer, these questions prompt us to consider sets of obstruction graphs. Our main motivation is: any small change to such an instance changes its minimum vertex cover size. Moreover, different elements of the same set F_k tend to differ substantially. So despite being finite, a set of obstructions tends to cover a wide range of graph structures that are mainly different types of problem instances. This simple idea appeared to be worth exploring, but we did not expect it to be as effective as it proved to be, as we shall see in the next section.

As mentioned earlier, designing an algorithm for computing all the elements of $\mathcal{O}b(k)$ is too difficult. We use the following methods described in [13] to construct a sufficient number of connected obstructions. The reason behind our use of connected obstructions stems from the fact that every connected component of an obstruction is an obstruction for a smaller value of k. Given a connected graph $G \in \mathcal{O}b(k)$, we produce a graph $G' \in \mathcal{O}b(k+1)$ by applying Algorithms 3 or 4 described below.

```
$V(G') \leftarrow V(G) \cup \{v', v''\}$ //$v', v''$ new vertices
$E(G') \leftarrow E(G) - \{v_1v_2\}$
$E(G') \leftarrow E(G') \cup \{v_1v', v'v'', v''v_2\}$
```

Algorithm 3 simply consists of subdividing an edge v_1v_2 twice, i.e., replacing it with a path of length three. Obviously, the minimum vertex cover size of the resulting instance is $k+1$, except when $k=0$. In this latter case there is exactly one obstruction, which is K_1, and the edge-subdivision method yields a path of length three (which is not an obstruction since deleting the middle edge does not decrease the minimum vertex cover size).

```
$V(G') \leftarrow V(G) \cup \{v'\}$ //$v'$ is a new vertex
$E(G') \leftarrow E(G) \cup \{v'v\} \cup \{v'u \mid u \in \mathcal{N}(v)\}$
```

Our strategy for computing obstructions is based on using Algorithms 3 and 4, starting from $\mathcal{O}b(3)$. However, we observed that the two methods can produce many duplicates, i.e., isomorphic graphs, which has two disadvantages. First, it would be prohibitive to compute obstructions for larger values of k, due to the already exponential growth in $|\mathcal{O}b(k)|$. Second, from a mere learning perspective, there is no merit in training a machine on essentially the same (relabeled) instance.

With the above note in mind, our method proceeds into filtering out all the produced duplicates. This additional step has enabled us to obtain a complete set of connected obstructions for $k \leq 10$, while a mere application of the methods from [13] prohibited us from going beyond $k=7$. To detect isomorphic instances, we used Nauty & Traces C-library [26]. As a byproduct we managed to generate (for the first time) accurate numbers of connected obstructions, as reported in Table 1 below.

5 Experiments

To properly assess the utility of our obstructions-based training approach, we adopted the same neural network and experimental setup of Dai et al. in [11]. In fact we used the same hyper-parameters, making sure we only change the training and validation data between the two approaches.

Table 1 Number of obtained connected obstructions for size-k vertex cover versus the counts obtained from [6, 12] and [14]

k	Our exact count	Previous count
4	8	8
5	26	31
6	124	188
7	728	1930
8	5118	unknown
9	39900	unknown
10	335537	unknown

Training the network starts by randomly initializing its weights, and at each iteration/epoch we pass the training data to the network to improve its weights and then evaluate the network on the validation data and finally take the best weights over all iterations.

The selected data sets consist of 19 widely used graphs from the Stanford Large Network Dataset Collection [25] and the Network Repository [29] as well as the meme-tracker data used by [11].

We compared the solution size produced by Dai's random subsets approach in [11] versus our obstructions' approach. Moreover, we compared both methods to the maximum-degree heuristic and the factor-two approximation algorithm, dubbed Alg1 and Alg2 in what follows, respectively. The graphs are listed in Table 2 along with the minimum vertex cover size of each, and we report our findings (i.e., computed vertex cover sizes) in Table 3.

As can be seen from Table 3, the two approaches adopted by Dai et al. and our work are comparable on almost all graphs. They both consistently outperform Alg2 and are nearly identical to Alg1, which is known to deliver near-optimum results in practice (despite its unbounded worst-case approximation ratio).

The comparison with the approach of Dai et al. is the most notable in the displayed results: despite the much smaller training set, the use of obstructions proved to be highly effective. In fact, the obstruction-based method can sometimes significantly improve on the previous method as in the example of the MANN-a45 graph. The results suggest that the obtained (or learned) vertex cover solver is comparable to the best-known heuristic. Overall, and based on the reported experimental results, the main advantage of using obstructions is two-fold:

1. Fast convergence (our method reached its best version in ≈ 3000 iterations while Dai's method used $\approx 700,000$).
2. Stability of learning, as shown in Fig. 1.

We present in Fig. 1 the mean squared error over time of each algorithm, taken over all the data instances, after each epoch of training over their respective training sets. To explicate, after each training iteration we computed:

$$\frac{1}{20}\sum_{G}(algorithm_i(G) - vc(G))^2$$

where $algorithm_i(G)$ is the vertex cover size computed by each of the four algorithms (Alg1, Alg2, the model trained without obstructions and the same model trained with obstructions). The sum is taken over all the 20 graphs shown in the above tables.

Table 2 The graphs used, along with the sizes of their optimum vertex covers

Graph	$\|V\|$	$\|E\|$	VC
C2000-5	2000	999164	1984
C250-9	250	3141	206
C500-9	500	12418	443
MANN-a27	378	702	252
MANN-a45	1035	1980	690
brock800-1	800	112095	777
c-fat200-1	200	18366	188
c-fat200-5	200	11427	142
gen400-p0-9-55	400	7980	345
gen400-p0-9-65	400	7980	335
hamming10-2	1024	5120	512
hamming8-4	256	11776	240
keller4	171	5100	160
p-hat1500-1	1500	839327	1488
p-hat700-2	700	122922	656
p-hat700-3	700	61640	638
san400-0-9-1	400	7980	300
sanr200-0-9	200	2037	158
sanr400-0-7	400	23931	379
Meme-tracker	960	4888	–

Table 3 Performance on real graphs: a comparison between the best performing network of [11] (after $\approx 700,000$ epochs) vs. our model after 3000 epochs

Graph	Alg1	Alg2	Random subsets	Obstructions method
C2000-5	1989	1996	1991	1991
C250-9	216	236	214	211
C500-9	453	484	456	453
MANN-a27	260	280	266	261
MANN-a45	704	740	740	705
brock800-1	785	796	788	785
c-fat200-1	187	198	188	192
c-fat200-5	141	192	142	142
gen400-p0-9-55	370	386	371	371
gen400-p0-9-65	367	384	367	368
hamming10-2	511	948	513	515
hamming8-4	239	252	240	240
keller4	162	166	163	163
p-hat1500-1	1490	1498	1493	1494
p-hat700-2	658	688	662	669
p-hat700-3	642	684	647	642
san400-0-9-1	349	386	350	350
sanr200-0-9	161	184	163	164
sanr400-0-7	382	394	386	384
Meme-tracker	482	658	480	481

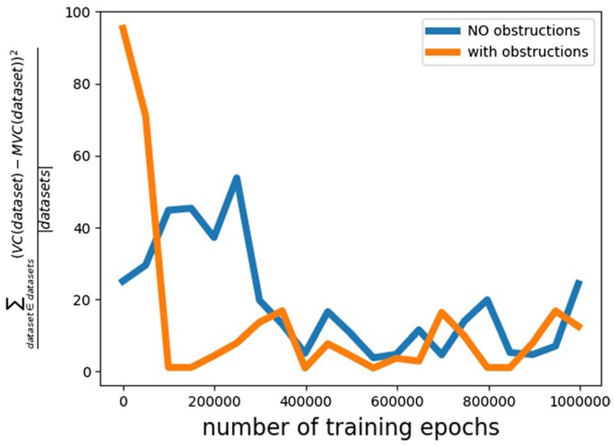

Fig. 1 Average MSE of the two models

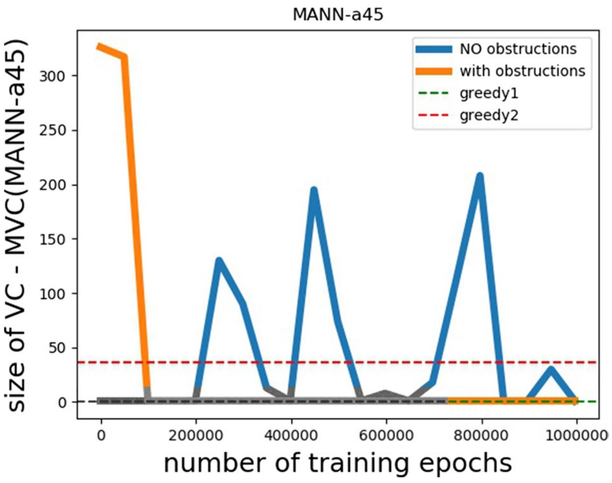

Fig. 2 The output of the two models over the MANN-a45 dataset

Finally, to further illustrate the effectiveness and fast convergence of the obstructions' approach, we display in Figs. 2, 3 and 4 the error over time for 3 of the 20 data sets chosen so as to summarize the average overall performance.

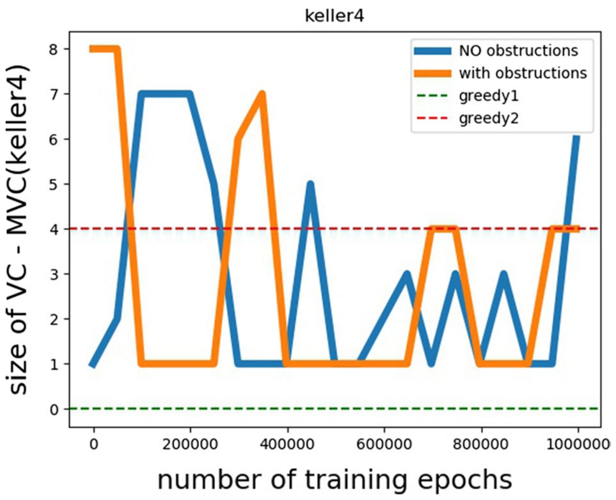

Fig. 3 The Keller4 dataset

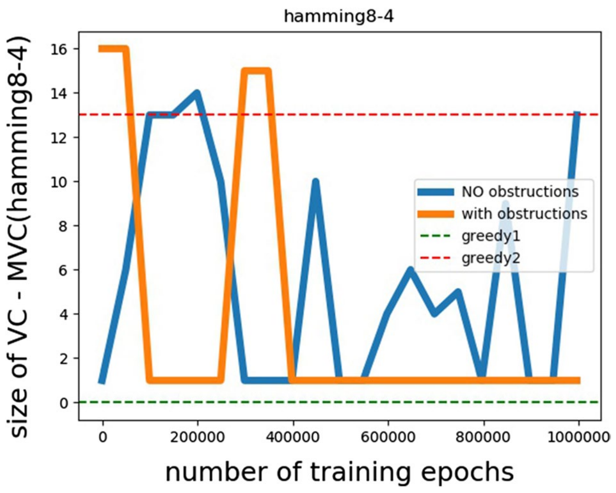

Fig. 4 The hamming8-4 dataset

6 Conclusion

In this paper we tried to combine methods from graph theory and machine learning in an attempt to achieve improved approximation techniques for an optimization problem, namely Vertex Cover. The main idea is to use the notion of an obstruction set for training. The reported results proved that using obstructions can have a tremendous impact on the training time, in our case reducing the number of iterations from ≈ 700000 to ≈ 3000 in the training time needed for convergence. Moreover, although we used only a small subset of vertex cover obstructions, we consistently obtained close to optimum solutions, and sometimes better than previously reported results, even at an early stage during training.

Our approach can possibly be applied to other minor-closed problems such as Feedback Vertex Set, Treewidth/Pathwidth, Cycle Packing, etc., provided some mechanisms for computing small obstructions are available or can be developed. We used Vertex Cover as a case study because among all the known minor-closed graph theoretic problems, Vertex

Cover is (so far) the only problem that seems to have well developed practical algorithms for computing obstructions. We believe this work can therefore initiate a number of similar projects on various minor-closed problems, hopefully reviving the work on developing techniques for computing respective obstruction sets.

An alternative possibly interesting approach would be to consider the use of "hard" problem instances instead of obstructions, especially for the problems that are not minor-closed. In fact, one might ideally think that learning how to compute solutions for hard instances may result in learning effective heuristic models. While this is worth exploring in future research, we should note that hardness in the exact optimization sense might not always mean hardness for heuristic optimization. For example, some of the hardest instances for Vertex Cover are regular graphs for which we can compute near-optimum solutions via fast heuristics and parameterized algorithms while computing the exact minimum takes a very long time (see Table 2 in [1]). We should note, finally, that obstructions are not necessarily hard instances. They have been chosen in our work because they are unique "extreme" (and diverse) instances in the sense that any slightest change that causes reduction in graph size flips a no-instance (for a certain vertex-cover size) to a yes-instance.

Funding No funding was received for conducting this study.

Data availability The data sets used in our experiments are publicly available. The graphs used in [11] can be obtained from http://www.memetracker.org/ and the authors' public github site (https://github.com/Hanjun-Dai/graphnn). The other graphs are obtained from the Stanford Large Network Dataset Collection [25] and the Network Repository [29].

Declarations

Conflicts of interests/Competing interests The authors have no competing interests or conflict of interest to declare that are relevant to the content of this article.

References

1. Abu-Khzam, F.N., Langston, M.A., Mouawad, A.E., Nolan, C.P.: A hybrid graph representation for recursive backtracking algorithms. In: Lee, D., Chen, D.Z., Ying, S. (eds.) Frontiers in Algorithmics, 4th International Workshop, FAW 2010, Wuhan, China, August 11-13, 2010. Proceedings, vol. 6213 of Lecture Notes in Computer Science, pp. 136–147. Springer (2010)
2. Abu-Khzam, F.N., Langston, M.A., Suters W.H.: Fast, effective vertex cover kernelization: a tale of two algorithms. In: 2005 ACS / IEEE International Conference on Computer Systems and Applications (AICCSA 2005), January 3-6, 2005, Cairo, Egypt, pp. 16. IEEE Computer Society (2005)
3. Adler, I., Grohe, M., Kreutzer, S.: Computing excluded minors (2008)
4. Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S.: Neural combinatorial optimization with reinforcement learning (2016)
5. Bruna, J., Zaremba, W., Szlam, A., LeCun, Y.: Spectral networks and locally connected networks on graphs (2013)
6. Cattell, K., Dinneen, M.J.: A characterization of graphs with vertex cover up to five. In: Bouchitté, V., Morvan, M. (eds.) Orders, Algorithms, and Applications, International Workshop ORDAL '94, Lyon, France, July 4-8, 1994, Proceedings, vol. 831 of Lecture Notes in Computer Science, pp. 86–99. Springer (1994)
7. Cattell, K., Dinneen, M.J., Downey, R.G., Fellows, M.R., Langston, M.A.: On computing graph minor obstruction sets. Theoret. Comput. Sci. **233**(1–2), 107–127 (2000)
8. Chlebík, M., Chlebíková, J.: Crown reductions for the minimum weighted vertex cover problem. Discret. Appl. Math. **156**(3), 292–312 (2008)

9. Courcelle, B., Downey, R.G., Fellows, M.R.: A note on the computability of graph minor obstruction sets for monadic second order ideals. J. Univers. Comput. Sci. **3**(11), 1194–1198 (1997)
10. Dai, H., Dai, B., Song, L.: Discriminative embeddings of latent variable models for structured data. In: Balcan, M., Weinberger, K.Q. (eds.) Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016, vol. 48 of JMLR Workshop and Conference Proceedings, pp. 2702–2711. JMLR.org (2016)
11. Dai, H., Kozareva, Z., Dai, B., Smola, A.J., Song, L.: Learning steady-states of iterative algorithms over graphs. In: Dy, J.G., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, vol. 80 of Proceedings of Machine Learning Research, pp. 1114–1122. PMLR (2018)
12. Dinneen, M., Versteegen, R.: Obstructions for the Graphs of Vertex Cover Seven. Technical report, Centre for Discrete Mathematics and Theoretical Computer Science (CDMTCS) (2012)
13. Dinneen, M.J., Lai, R.: Properties of vertex cover obstructions. Discret. Math. **307**(21), 2484–2500 (2007)
14. Dinneen, M.J., Xiong, L.: Minor-order obstructions for the graphs of vertex cover 6. Journal of Graph Theory **41**(3), 163–178 (2002)
15. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Monographs in Computer Science. Springer (1999)
16. Fellows, M.R., Langston, M.A.: Nonconstructive tools for proving polynomial-time decidability. J. ACM **35**(3), 727–739 (1988)
17. Fellows, M.R., Langston, M.A.: On search, decision, and the efficiency of polynomial-time algorithms. J. Comput. Syst. Sci. **49**(3), 769–779 (1994)
18. Frasconi, P., Gori, M., Sperduti, A.: A general framework for adaptive processing of data structures. IEEE Trans. Neural Netw. **9**(5), 768–786 (1998)
19. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H Freeman (1979)
20. Gori, M., Monfardini, G., Scarselli, F.: A new model for learning in graph domains. Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005., 2:729–734 vol. 2 (2005)
21. Karp, R.M.: Reducibility among Combinatorial Problems, pp. 85–103. Springer, US, Boston, MA (1972)
22. Khot, S., Regev, O.: Vertex cover might be hard to approximate to within 2 - ϵ. J. Comput. Syst. Sci. **74**(3), 335–349 (2008). Computational Complexity 2003
23. Kinnersley, N.G., Langston, M.A.: obstruction set isolation for the gate matrix layout problem. Discret. Appl. Math. **54**(2–3), 169–213 (1994)
24. Kipf, T.N., Welling, M.: Variational graph auto-encoders (2016)
25. Leskovec, J., Krevl, A.: SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data (2014)
26. McKay, B.D., Piperno, A.: Practical graph isomorphism, { II }. Journal of Symbolic Computation **60**, 94–112 (2014)
27. Nemhauser, G.L., L.E.T, Jr.: Vertex packings: Structural properties and algorithms. Math. Program. **8**(1), 232–248 (1975)
28. Robertson, N., Seymour, P.: Graph minors. xx. wagner's conjecture. J. Comb. Theory, Ser. B **92**(2), 325–357 (2004). Special Issue Dedicated to Professor W.T. Tutte
29. Rossi, R.A., Ahmed, N.K.: An interactive data repository with visual analytics. SIGKDD Explor. **17**(2), 37–41 (2016)
30. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Trans. Neural Netw. **20**(1), 61–80 (2009)
31. Vinyals, O., Fortunato, M., Jaitly, N.: Pointer networks. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems 28, pp. 2692–2700. Curran Associates, Inc. (2015)
32. Wagner, K.: Über eine eigenschaft der ebenen komplexe. Math. Ann. **114**(1), 570–590 (1937)
33. Zhang, Z., Cui, P., Zhu, W.: Deep learning on graphs: A survey. arXiv:1812.04202 (2018)