DEEP AUTO-DEFERRING POLICY FOR COMBINATORIAL OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Designing efficient algorithms for combinatorial optimization appears ubiquitously in various scientific fields. Recently, deep reinforcement learning (DRL) frameworks have gained considerable attention as a new approach: they can automatically learn the design of a good solver without using any sophisticated knowledge or hand-crafted heuristic specialized for the target problem. However, the number of stages (until reaching the final solution) required by existing DRL solvers is proportional to the size of the input graph, which hurts their scalability to large-scale instances. In this paper, we seek to resolve this issue by proposing a novel design of DRL's policy, coined auto-deferring policy (ADP), automatically stretching or shrinking its decision process. Specifically, it decides whether to finalize the value of each vertex at the current stage or defer to determine it at later stages. We apply the proposed ADP framework to the maximum independent set (MIS) problem, a prototype of NP-complete problems, under various scenarios. Our experimental results demonstrate significant improvement of ADP over the current state-of-the-art DRL scheme in terms of computational efficiency and approximation quality. The reported performance of our generic DRL scheme is also comparable with that of the state-of-the-art solvers specialized for MIS, e.g., ADP outperforms them for some graphs with millions of vertices.

1 INTRODUCTION

Combinatorial optimization is an important mathematical field addressing fundamental questions of computation, where its popular examples include the maximum independent set (MIS, Miller & Muller 1960), satisfiability (SAT, Schaefer 1978) and traveling salesman problem (TSP, Voigt 1831). Such problems also arise in various applied fields, e.g., sociology (Harary & Ross, 1957), operations research (Feo et al., 1994) and bioinformatics (Gardiner et al., 2000). However, most combinatorial optimization problems are NP-hard to solve, i.e., exact solutions are typically intractable to find in practical situations. To alleviate this issue, there have been huge efforts in designing fast heuristic solvers (Biere et al., 2009; Knuth, 1997; Mezard et al., 2009) that generate approximate solutions for such scenarios.

Recently, the remarkable progress in deep learning has stimulated increased interest in learning such heuristics based on deep neural networks (DNNs). Such learning-based approaches are attractive since one could obtain an efficient approximation algorithm for any domains of a problem without sophisticated knowledge. As the most straight-forward way, supervised learning schemes can be used for training DNNs to imitate the solutions obtained from existing solvers (Vinyals et al., 2015; Li et al., 2018; Selsam et al., 2019). However, such a direction can be criticized, for its quality and applicability are bounded by those of existing solvers. An ideal direction is to discover new solutions in a fully unsupervised manner, potentially outperforming those based on domain-specific knowledge.

To this end, deep reinforcement learning (DRL) schemes have been studied in the literature (Bello et al., 2016; Khalil et al., 2017; Deudon et al., 2018; Kool et al., 2019) as a Markov decision process (MDP) can be naturally designed with rewards derived from the optimization objective of the target problem. Then, the corresponding agent can be trained (without any human guidance) based on existing training schemes of DRL, e.g., Bello et al. (2016) trained the so-called pointer network for the TSP based on actor-critic training. Although conceptually appealing, the existing DRL-based

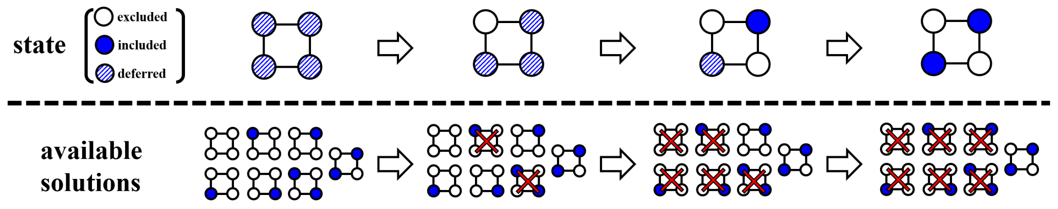


Figure 1: Illustration of the proposed Markov decision process.

methods struggle to compete with the existing highly optimized solvers. In particular, the gap grows larger when the problem requires solutions of higher dimension or more complex structure.

Our motivation stems from the observation that existing DRL-based solvers lack efficient policies for generating solutions to combinatorial problems. Specifically, they are mostly based on emulating greedy iterative heuristics (Bello et al., 2016; Khalil et al., 2017) and become too slow for training on large graphs. Their choice seems inevitable since an algorithm that generates a solution based on a single feed-forward pass of DNN is potentially hard to train due to large variance in reward signals coming from high dimensional solutions.

Contribution. In this paper, we propose a new scalable DRL framework, coined auto-deferring policy (ADP), for solving combinatorial problems on large graphs. Although the proposed ADP can be applied to any combinatorial optimization tasks, we focus on the popular MIS problem which attempts to find a maximum set of vertices in the graph where no pair of vertices are adjacent to each other. Our choice of the MIS problem is motivated by its hardness and applicability. First, the MIS problem is impossible to approximate in polynomial time by a constant factor (Hastad, 1996), in contrast to (Euclidean or metric) TSP which can be approximated by factor of 1.5 (Christofides, 1976). Next, it has wide applications including classification theory (Feo et al., 1994), computer vision (Sander et al., 2008) and communication algorithms (Jiang & Walrand, 2010).

The main novelty of ADP is automatically stretching the determination of the solution throughout multiple steps. In particular, the agent iteratively acts on every undetermined vertex for either (a) determining the membership of the vertex in the solution or (b) deferring the determination to be made in later steps (see Figure 1 for illustration). Inspired by the celebrated survey propagation (Braunstein et al., 2005) for solving the SAT problem, ADP could be interpreted as prioritizing the "easier" decisions to be made first, which in turn simplifies the harder ones by eliminating the source of uncertainties. Compared to the greedy strategy (Khalil et al., 2017) which determines the membership of a single vertex at each step, our framework brings significant speedup by allowing determinations on as many vertices as possible to happen at once.

Based on such speedup, ADP can solve the optimization problem by generating a large number of candidate solutions in a limited time budget, then reporting the best solution among them. In such a scenario, it is beneficial for the algorithm to generate diverse candidates. To this end, we additionally give a novel diversification bonus to our agent during training, which explicitly encourages the agent to generate a large variety of solutions. Specifically, we create a "coupling" of Markov decision processes to generate two solutions for the given MIS problem and reward the agents for a large deviation between the solutions. The resulting reward efficiently stimulates the agent to explore high-dimensional input spaces and to improve the performance at the evaluation.

We empirically validate the ADP method on various types of graphs including the Erdös-Rényi (Erdős & Rényi, 1960) model, the Barabási-Albert (Albert & Barabási, 2002) model, the SATLIB (Hoos & Stützle, 2000) benchmark and real-world graphs (Hamilton et al., 2017; Yanardag & Vishwanathan, 2015; Leskovec & Sosič, 2016). Our algorithm shows consistent superiority over the existing state-of-the-art DRL method (Khalil et al., 2017) both in terms of speed and quality of the solution, and can compete with the state-of-the-art MIS heuristic (ReduMIS, Lamm et al. 2017) under a similar time budget. For example, ADP even outperforms ReduMIS in the Barabási-Albert graph with two million vertices using a smaller amount of time. Furthermore, we also show that our fully learning-based algorithm generalizes well even to graph types unseen during training. This sheds light on its potential of being a generic solver that works for arbitrary large-scale graphs.

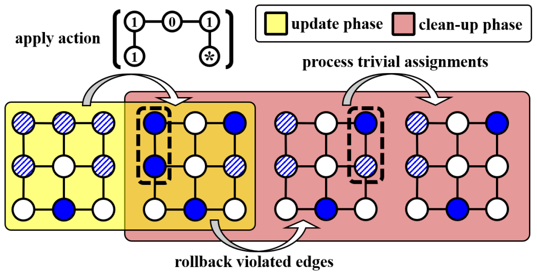


Figure 2: Illustration of the transition function with the update and the clean-up phases.

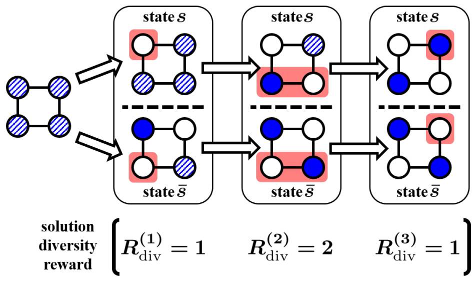

state s
state $\bar{s}$
solution diversity reward
$R^{(1)}_{\text{div}} = 1$
$R^{(2)}_{\text{div}} = 2$
$R^{(3)}_{\text{div}} = 1$

Figure 3: Illustration of coupled MDP with the corresponding solution diversity reward.

2 Deep auto-deferring policy

In this paper, we focus on solving the maximum independent set (MIS) problem. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertices $\mathcal{V}$ and edges $\mathcal{E}$, an *independent set* is a subset of vertices $\mathcal{I} \subseteq \mathcal{V}$ where no two vertices in the subset are adjacent to each other. A solution to the MIS problem can be represented as a binary vector $\boldsymbol{x} = [x_i : i \in \mathcal{V}] \in \{0,1\}^{\mathcal{V}}$ with maximum possible cardinality $\sum_{i \in \mathcal{V}} x_i$, where each element x_i indicates the membership of vertex i in the independent set $\mathcal{I}$, i.e., $x_i = 1$ if and only if $i \in \mathcal{I}$. Initially, the algorithm has no assumption about its output, i.e., both $x_i = 0$ and $x_i = 1$ are possible for all $i \in \mathcal{V}$. At each iteration, the agent acts on each undetermined vertex i by either (a) determining its membership to be a certain value, i.e., set $x_i = 0$ or $x_i = 1$, or (b) deferring the determination to be made later iterations. The iterations are repeated until all the membership of vertices in the independent set are determined. Such a strategy could be interpreted as progressively narrowing down the set of candidate solutions at each iteration (See Figure 1 for illustration).

2.1 Deferred Markov decision process

To train the agent via reinforcement learning, we formulate the proposed algorithm as a Markov decision process (MDP).

State. Each state of the MDP is represented as a *vertex-state* vector $\boldsymbol{s} = [s_i : i \in \mathcal{V}] \in \{0,1,*\}^{\mathcal{V}}$, where the vertex $i \in \mathcal{V}$ is determined to be excluded or included in the independent set whenever $s_i = 0$ or $s_i = 1$, respectively. Otherwise, $s_i = *$ indicates the determination has been deferred and expected to be made in later iterations. The MDP is initialized with the deferred vertex-states, i.e., $s_i = *$ for all $i \in \mathcal{V}$, and terminated when there is no deferred vertex-state left.

Action. Actions corresponds to new assignments for the next state of vertices. Since vertex-states of included and excluded vertices are immutable, the assignments are defined only on the deferred vertices. It is represented as a vector $\boldsymbol{a}_* = [a_i : i \in \mathcal{V}_*] \in \{0,1,*\}^{\mathcal{V}_*}$ where $\mathcal{V}_*$ denotes a set of current deferred vertices, i.e., $\mathcal{V}_* = \{i : i \in \mathcal{V}, x_i = *\}$.

Transition. Given two consecutive states $\boldsymbol{s}, \boldsymbol{s}'$ and the corresponding assignment $\boldsymbol{a}_*$, the transition $P_{\boldsymbol{a}_*}(\boldsymbol{s}, \boldsymbol{s}')$ consists of two deterministic phases: the *update phase* and the *clean-up phase*. In the update phase, the assignment $\boldsymbol{a}_*$ generated by the policy is updated for the corresponding vertices $\mathcal{V}_*$ to result in an intermediate vertex-state $\widehat{\boldsymbol{s}}$, i.e., $\widehat{s}_i = a_i$ if $i \in \mathcal{V}_*$ and $\widehat{s}_i = s_i$ otherwise. In the cleanup phase, the intermediate vertex-state vector $\widehat{\boldsymbol{s}}$ is modified to yield a valid vertex-state vector $\boldsymbol{s}'$, where included vertices are only adjacent to the excluded vertices. To this end, whenever there exists a pair of included vertices adjacent to each other, they are both mapped back to the deferred vertex-state. Next, any deferred vertex neighboring with an included vertex is excluded. If the state reaches the pre-defined time limit, all deferred vertices are automatically excluded. See Figure 2 for a more detailed illustration of the transition.

Reward. Finally, reward $R(\boldsymbol{s}, \boldsymbol{s}')$ is defined as the increase in cardinality of included vertices, i.e., $R(\boldsymbol{s}, \boldsymbol{s}') = \sum_{i \in \mathcal{V}_* \setminus \mathcal{V}'_*} s'_i$, where $\mathcal{V}_*$ and $\mathcal{V}'_*$ are the set of vertices with deferred vertex-state with respect to $\boldsymbol{s}$ and $\boldsymbol{s}'$, respectively. By doing so, the overall return of the MDP corresponds to the cardinality of the independent set returned by our algorithm.

2.2 Training with diversification reward

Next, we introduce an additional reward term for encouraging diversification of solutions generated by the agent. Such regularization is motivated by our evaluation method which samples multiple candidate solutions in order to report the best one as the final output. For such scenarios, it would be beneficial to generate diverse solutions of high maximum score, rather than ones of similar scores. One might argue that the existing entropy regularization (Williams & Peng, 1991) for encouraging exploration over MDP could be used for this purpose. However, the entropy regularization attempts to generate diverse trajectories of the same MDP which does not necessarily lead to diverse solutions at last, since there exist many trajectories resulting in the same solution (see Section 2.1). We instead directly maximize the diversity among solutions by a new reward term. To this end, we "couple" two copies of MDPs defined in Section 2.1 into a new MDP by sharing the same graph $\mathcal{G}$ with a pair of distinct vertex-state vectors $(\boldsymbol{s}, \bar{\boldsymbol{s}})$. Although the coupled MDPs are defined on the same graph, the corresponding agents work independently to result in a pair of solutions $(\boldsymbol{x}, \bar{\boldsymbol{x}})$. Then, we directly reward the deviation between the coupled solutions in terms of ℓ_1-norm, i.e., $\|\boldsymbol{x} - \bar{\boldsymbol{x}}\|_1$. Similar to the original objective of MIS, it is decomposed into rewards in each iteration of the MDP defined as follows:

$$R_{\text{div}}(\boldsymbol{s}, \boldsymbol{s}', \bar{\boldsymbol{s}}, \bar{\boldsymbol{s}}') = \sum_{i \in \widehat{\mathcal{V}}} |s_i' - \bar{s}_i'|, \quad \text{where} \quad \widehat{\mathcal{V}} = (\mathcal{V}_* \setminus \mathcal{V}_*') \cup (\bar{\mathcal{V}}_* \setminus \bar{\mathcal{V}}_*'),$$

where $(\boldsymbol{s}', \bar{\boldsymbol{s}}')$ denotes the next pair of vertex-states in the coupled MDP. One can observe that $\widehat{\mathcal{V}}$ denotes the most recently updated vertices in each MDP. In practice, such reward R_{div} can be used along with the maximum entropy regularization for training the agent to achieve the best performance. See Figure 3 for an example of coupled MDP with the proposed reward.

Our algorithm is based on actor-critic training with policy network $\pi_\theta(\boldsymbol{a}|\boldsymbol{s})$ and value network $V_\phi(\boldsymbol{s})$ parameterized by the graph convolutional network (GCN, Kipf & Welling 2017). Each GCN consists of multiple layers h_n with $n = 1, \cdots, N$ where the n-th layer with weights $\mathbf{W}_1^{(n)}$ and $\mathbf{W}_2^{(n)}$ performs the following transformation on input $\mathbf{H}$:

$$h^{(n)}(\mathbf{H}) = \text{ReLU}\big(\mathbf{H}\mathbf{W}_1^{(n)} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{H}\mathbf{W}_2^{(n)}\big).$$

Here $\mathbf{A}, \mathbf{D}$ correspond to adjacency and degree matrix of the graph $\mathcal{G}$, respectively. At the final layer, the policy and value networks apply softmax function and graph readout function with sum pooling (Xu et al., 2019) instead of ReLU to generate actions and value estimates, respectively. We only consider the subgraph that is induced on the deferred vertices $\mathcal{V}_*$ as the input of the networks since the determined part of the graph no longer affects future rewards of the MDP. Features corresponding to the vertices are given as their node degrees and the current iteration-index of MDP.

In order to train the agent, proximal policy optimization (Schulman et al., 2017) is used. Specifically, networks are trained for maximizing the following objective:

$$\mathcal{L} := \mathbb{E}_t \Bigg[\min \Bigg(\widehat{A}(\boldsymbol{s}^{(t)}) \prod_{i \in \mathcal{V}} r_i^{(t)}(\theta), \widehat{A}(\boldsymbol{s}^{(t)}) \prod_{i \in \mathcal{V}} \texttt{clip}(r_i^{(t)}(\theta), 1 - \varepsilon, 1 + \varepsilon) \Bigg) \Bigg],$$

$$r_i^{(t)}(\theta) = \frac{\pi_\theta(a_i^{(t)}|\boldsymbol{s}^{(t)})}{\pi_{\theta_{\text{old}}}(a_i^{(t)}|\boldsymbol{s}^{(t)})}, \qquad \widehat{A}(s) = \sum_{t'=t}^{T} r^{(t')} - V_\phi(s),$$

where $\boldsymbol{s}^{(t)}$ denotes the t-th vertex-state vector and other elements of the MDP are defined similarly. In addition, $\texttt{clip}(\cdot)$ is the clipping function for updating the agent more conservatively and θ_{old} is the parameter of the policy network from the previous iteration of updates.

3 Experiments

In this section, we report experimental results on the proposed auto-deferring policy (ADP) described in Section 2. To this end, experiments were conducted on a large range of graphs varying from small synthetic graphs to large-scale real-world graphs. We evaluated our ADP scheme by sampling multiple solutions and then reporting the performance of the best solution. The resulting schemes are coined ADP-10, ADP-100 and ADP-1000 corresponding to the number of samples chosen from 10, 100 and 1000, respectively. We compared our framework with the deep reinforcement learning

Table 1: Performance evaluation on ER and BA datasets. The bold numbers indicate the best performance within the same category of algorithms. The relative differences shown in brackets are measured with respect to S2V-DQN.

			Traditional			DRL-based			
Type	$\lvert\mathcal{V}\rvert$	Value	ES	CPLEX	ReduMIS	S2V-DQN	ADP-10	ADP-100	ADP-1000
ER	(15, 20)	Obj.	7.800	**8.844**	**8.844**	8.840	**8.844 (+0.1%)**	**8.844 (+0.1%)**	**8.844 (+0.1%)**
		Time	0.005	0.003	0.024	0.004	0.002 (−60.7%)	0.005 (+15.6%)	0.056 (+1185.5%)
	(40, 50)	Obj.	13.83	**16.57**	**16.57**	16.42	16.55 (+0.7%)	**16.57 (+0.9%)**	**16.57 (+0.9%)**
		Time	0.033	0.062	12.374	0.015	0.004 (−73.1%)	0.016 (+9.6%)	0.164 (+1002.0%)
	(50, 100)	Obj.	17.01	**21.11**	**21.11**	20.61	21.04 (+2.1%)	21.10 (+2.4%)	**21.11 (+2.4%)**
		Time	0.117	0.137	24.387	0.030	0.007 (−77.8%)	0.028 (−4.3%)	0.283 (+852.6%)
	(100, 200)	Obj.	21.59	27.87	**27.95**	26.27	27.67 (+5.3%)	27.87 (+6.1%)	**27.93 (+6.4%)**
		Time	0.608	10.748	30.109	0.078	0.029 (−63.5%)	0.085 (+8.5%)	0.666 (+748.7%)
	(400, 500)	Obj.	29.28	35.76	**39.83**	35.05	38.29 (+9.2%)	39.11 (+11.6%)	**39.54 (+12.8%)**
		Time	10.823	30.085	30.432	0.633	0.158 (−75.1%)	0.407 (−35.7%)	2.768 (+336.8%)
BA	(15, 20)	Obj.	6.06	**7.019**	**7.019**	7.011	**7.019 (+0.1%)**	**7.019 (+0.1%)**	**7.019 (+0.1%)**
		Time	0.003	0.004	0.041	0.005	0.002 (−60.2%)	0.006 (+19.4)	0.064 (+1103.4)
	(40, 50)	Obj.	14.81	**18.91**	**18.91**	18.87	**18.91 (+0.2%)**	**18.91 (+0.2%)**	**18.91 (+0.2%)**
		Time	0.031	0.020	0.396	0.013	0.003 (−79.2%)	0.011 (−18.4%)	0.118 (+778.1%)
	(50, 100)	Obj.	24.77	**32.07**	**32.07**	31.96	**32.07 (+0.3%)**	**32.07 (+0.3%)**	**32.07 (+0.3%)**
		Time	0.130	0.038	0.739	0.022	0.003 (−84.7%)	0.018 (−19.0%)	0.199 (+794.3%)
	(100, 200)	Obj.	49.87	**66.07**	**66.07**	65.52	66.05 (+0.8%)	**66.07 (+0.8%)**	**66.07 (+0.8%)**
		Time	0.938	0.088	2.417	0.047	0.007 (−85.2%)	0.038 (−19.6%)	0.380 (+703.6%)
	(400, 500)	Obj.	148.51	**204.14**	**204.14**	202.91	204.04 (+0.6%)	204.10 (+0.6%)	**204.12** (+0.6)%
		Time	23.277	0.322	15.080	0.177	0.024 (−86.4%)	0.131 (−25.8%)	1.111 (+527.5%)

Table 2: Performance evaluation on SATLIB, PPI, REDDIT and as-Caida datasets. The bold numbers indicate the best performance within the same category of algorithms. The relative differences shown in brackets are measured with respect to S2V-DQN, except for the case of as-Caida dataset where S2V-DQN underperforms significantly.[1]

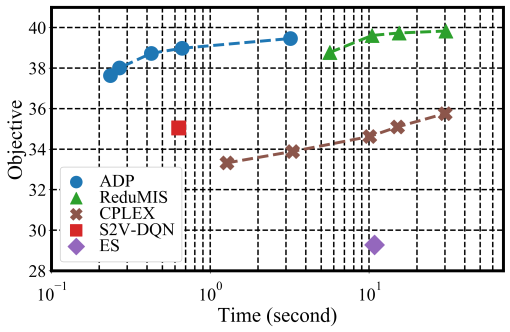
			Traditional		DRL-based			
Type	$\lvert\mathcal{V}\rvert$	Value	CPLEX	ReduMIS	S2V-DQN	ADP-10	ADP-100	ADP-1000
SATLIB	(1209, 1347)	Obj.	426.8	**426.9**	413.8	423.8 (+2.4%)	424.8 (+2.7%)	**425.4 (+2.8%)**
		Time	9.490	30.110	2.260	0.311 (−86.2%)	1.830 (−19.0%)	16.409 (+626.1%)
PPI	(591, 3480)	Obj.	**1147.5**	**1147.5**	893.0	1144.5 (+28.2%)	1146.5 (+28.4%)	**1147.0 (+28.4%)**
		Time	24.685	30.23	6.285	0.786 (−87.5%)	1.770 (−71.8%)	11.469 (+82.3%)
REDDIT (MULTI-5K)	(22, 3648)	Obj.	**370.6**	**370.6**	370.1	**370.6 (+0.1%)**	**370.6 (+0.1%)**	**370.6 (+0.1%)**
		Time	0.008	0.159	0.076	0.071 (−6.6%)	0.551 (+625.0%)	5.500 (+7136.8%)
REDDIT (MULTI-12K)	(2, 3782)	Obj.	**303.5**	**303.5**	302.8	257.4 (−15.0%)	292.6 (−3.4%)	**303.5 (+0.2%)**
		Time	0.007	0.188	1.883	0.003 (−99.8%)	0.025 (−98.7%)	0.451 (−76.1%)
REDDIT (BINARY)	(6, 3782)	Obj.	**329.3**	**329.3**	328.6	**329.3 (+0.2%)**	**329.3 (+0.2%)**	**329.3 (+0.2%)**
		Time	0.007	0.306	0.055	0.020 (−63.6%)	0.173 (+214.6%)	2.627 (+4676.4%)
as-Caida	(8020, 26 475)	Obj.	**20 049.2**	**20 049.2**	324.0	**20 049.2**	**20 049.2**	**20 049.2**
		Time	0.477	1.719	601.351	0.812	6.106	62.286

(DRL) algorithm by Khalil et al. (2017), coined S2V-DQN, for solving the MIS problem. Note that other DRL schemes in the literature, e.g., pointer network (Bello et al., 2016), and attention layer (Kool et al., 2019) are not comparable since they are specialized to TSP-like problems. We additionally consider three conventional MIS solvers as baselines. First, we consider the theoretically guaranteed algorithm of Boppana & Halldórsson (1992) based on iterative exclusion of subgraphs, coined ES, having approximation ratio $O(|\mathcal{V}|/(\log|\mathcal{V}|)^2)$ for the MIS problem. Next, we consider the integer programming solver IBM ILOG CPLEX Optimization Studio V12.9.0 (ILO, 2014), coined CPLEX. Finally, we also consider the heuristic proposed by Lamm et al. (2017), coined ReduMIS, which reported state-of-the-art performance on the MIS problem. Details on the implementation of the algorithms are provided in the appendix.

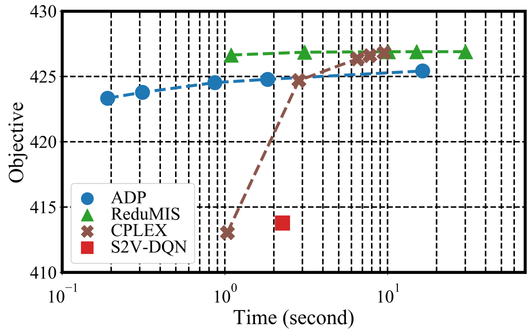
3.1 Performance evaluation

We now demonstrate the performance of our algorithm along with other baselines on various datasets. First, we consider experiments on randomly generated synthetic graphs from the Erdös-Rényi (ER, Erdős & Rényi 1960) and Barabási-Albert (BA, Albert & Barabási 2002) models. Following Khalil

[1] The authors of S2V-DQN only reported experiments with respect to graphs of size up to five hundred.



(a) ER-$(400, 500)$ dataset



(b) SATLIB dataset

Figure 4: Evaluation of trade-off between time and objective (upper-left side is of better trade-off).

et al. (2017), the edge ratio of ER graphs and average degree of BA graphs are set to 0.15 and 8, respectively. Datasets are specified by their type of model and an interval for choosing the number of vertices uniformly at random, e.g., ER-$(50, 100)$ denotes the set of ER graphs generated with the number of vertices larger than 50 and smaller than 100. Next, we consider experiments on more challenging datasets with larger sizes, namely the SATLIB, PPI, REDDIT, and as-Caida datasets constructed from SATLIB benchmark (Hoos & Stützle, 2000), protein-protein interactions (Hamilton et al., 2017), social networks (Yanardag & Vishwanathan, 2015) and road networks (Leskovec & Sosič, 2016), respectively. See the appendix for more details on the datasets. The time limit of the CPLEX and ReduMIS are set to 30 seconds on ER and BA datasets and 1800 seconds on the rest of the datasets in order to provide comparable baselines.[2] The corresponding results are reported in Table 1 and 2. Note that the ES method was excluded from comparison in large graphs since it required a prohibitive amount of computation.

In Table 1 and 2, one can observe that our ADP algorithms significantly outperform the deep reinforcement learning baseline, i.e., S2V-DQN, across all types of graphs and datasets. For example, ADP-10 is always able to find a better solution than S2V-DQN much faster. The gap grows larger in more challenging datasets, e.g., see Table 2. It is also impressive to observe that our algorithm is able to find the best solution in seven out of ten datasets in Table 1 and four out of five datasets in Table 2. Furthermore, we observe that our algorithm outperforms the CPLEX solver on ER-$(100, 200)$ and ER-$(400, 500)$ datasets, while consuming a smaller amount of time. The highly optimized ReduMIS solver tends to acquire the best solutions consistently under the maximum time limit we set. However, it is often worse than ours given a limited time budget, as described in what follows.

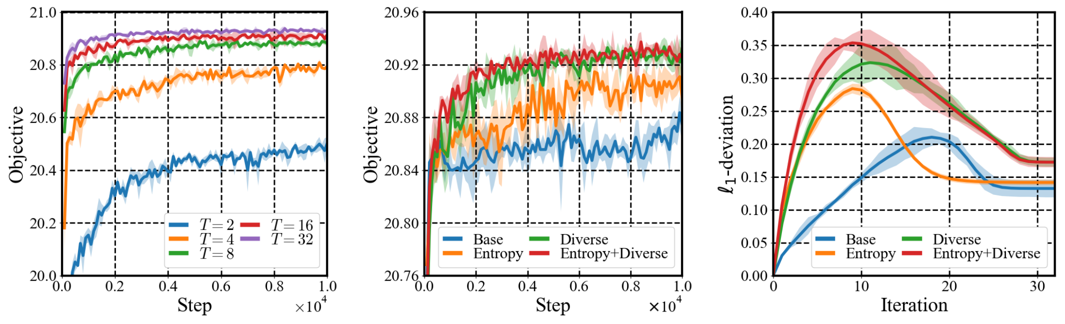
We investigate the trade-off between objective and time for algorithms in Figure 4. To this end, we evaluate algorithms on ER-$(400, 500)$ and SATLIB datasets with varying number of samples for ADP and time limits for ReduMIS and CPLEX. It is remarkable to observe that ADP outperforms the CPLEX solver on both datasets under reasonably limited time. Furthermore, for time limits smaller than 10 seconds, ADP outperforms ReduMIS on ER-$(400, 500)$ dataset.

3.2 Ablation Study

We ablate each component of our algorithm to validate its effectiveness. We first show that "stretching" the determination with deferred MDP indeed helps for solving the MIS problem. Specifically, we experiment with varying the maximum number of iterations T in MDP by $T \in \{2, 4, 8, 16, 32\}$ on ER-$(50, 100)$ dataset. Figure 5a reports the training curve for validation score of the corresponding experiments. We observe that the performance of ADP improves whenever an agent is given more time to generate the final solution, which verifies our claim that the choice of deferring the decision plays a crucial role in solving the MIS problem.

Next, we inspect the contribution of the solution diversity reward used in our algorithm. To this end, we trained agents with four options: (a) without any exploration bonus, coined Base, (b) with the conventional entropy bonus (Williams & Peng, 1991), coined Entropy, (c) with the proposed

[2] The solvers occasionally violate the time limit due to their pre-solving process.



(a) Performance with varying T (b) Contribution of each regularizers (c) Deviation in intermediate stages

Figure 5: Illustration of ablation studies done on ER-$(50, 100)$ dataset. The solid line and shaded regions represent the mean and standard deviation across 3 runs respectively Note that the standard deviation in (c) was enlarged ten times for better visibility.

Table 3: Generalization performance of ADP-1000 across synthetic graphs with varying types and sizes. Rows and columns correspond to datasets used for training and evaluating the model, respectively.

		ER				BA			
Type	$\lvert\mathcal{V}\rvert$	(40, 50)	(50, 100)	(100, 200)	(400, 500)	(40, 50)	(50, 100)	(100, 200)	(400, 500)
ER	(15, 20)	**16.57**	21.06	26.50	30.36	**18.91**	**32.07**	66.00	200.96
	(40, 50)	**16.57**	**21.11**	27.88	36.95	**18.91**	**32.07**	**66.07**	204.09
	(50, 100)	-	**21.11**	**27.94**	38.20	-	**32.07**	**66.07**	204.07
	(100, 200)	-	-	27.93	39.39	-	-	**66.07**	204.08
	(400, 500)	-	-	-	**39.54**	-	-	-	203.90
BA	(15, 20)	**16.57**	21.10	27.28	33.87	**18.91**	**32.07**	66.05	202.47
	(40, 50)	**16.57**	**21.11**	27.89	37.01	**18.91**	**32.07**	**66.07**	204.03
	(50, 100)	-	**21.11**	27.89	37.01	-	**32.07**	**66.07**	204.11
	(100, 200)	-	-	27.86	36.52	-	-	**66.07**	204.11
	(400, 500)	-	-	-	35.31	-	-	-	**204.12**

diversification bonus, coined Diverse, and (d) with both of the bonuses, coined Entropy+Diverse. The corresponding training curves for validation scores are reported in Figure 5b. We observe that the agent trained with the proposed diversification bonus outperforms other agents in terms of validation score, confirming the effectiveness of our proposed reward. One can also observe that both methods can be combined to yield better performance, i.e., Entropy+Diverse.

Finally, we further verify our claim that the maximum entropy regularization fails to capture the diversity of solutions effectively, while the proposed solution diversity reward term does. To this end, we compare the fore-mentioned agents with respect to the ℓ_1-deviations between the coupled intermediate vertex-states $\boldsymbol{s}$ and $\bar{\boldsymbol{s}}$, defined as $|\{i : i \in \mathcal{V}, s_i \neq \bar{s}_i\}|$. The corresponding results are shown in Figure 5c. Indeed, one can observe that the entropy regularization leads to large deviations during the intermediate stages, but converges to solutions with smaller deviations. On the contrary, agents trained on diversification rewards succeed in enlarging the deviation between the final solutions.

3.3 Generalization to Unseen Graphs

Now we examine the potential of our method as a generic solver, i.e., whether the algorithm's performance generalizes well to graphs unseen during training. To this end, we first construct experiments on generalization between ER and BA graphs, where we evaluate our method on different types and sizes of graphs from the training dataset. As shown in Table 3, our algorithm generalizes excellently across different sizes of graphs, e.g., the model trained on BA-$(50, 100)$ dataset achieves the best performance even on BA-$(400, 500)$ dataset. On the other side, models evaluated on unseen type of graphs tend to work slightly worse as expected. However, the results

Table 4: Performance evaluation for large-scale graphs. Out of budget (OB) is marked for runs violating the time and the memory budget of 10 000 seconds and 32 GB RAM, respectively. The bold numbers indicate the best performance within the same category of algorithms. The relative differences shown in brackets are measured with respect to S2V-DQN.

			Traditional		DRL-based			
Type	$\|\mathcal{V}\|$	Value	CPLEX	ReduMIS	S2V-DQN	ADP-10	ADP-100	ADP-1000
BA	1 000 000	Obj.	434 896	**457 349**	OB	457 753	**457 772**	OB
		Time	5967.82	1802.35		324.02	5112.25	
	2 000 000	Obj.	OB	**909 988**	OB	915 553	**915 573**	OB
		Time		4276.43		772.18	7662.87	
Citation (Cora)	2708	Obj.	**1451**	**1451**	1393	**1451** (**+4.2%**)	**1451** (**+4.2%**)	**1451** (**+4.2%**)
		Time	0.08	0.04	2.57	1.63 (−36.5%)	2.71 (+5.3%)	13.82 (+437.7%)
Citation (Citeseer)	3327	Obj.	**1867**	**1867**	1840	**1867** (**+1.5%**)	**1867** (**+1.5%**)	**1867** (**+1.5%**)
		Time	0.08	0.03	3.07	1.80 (−41.4%)	2.74 (−10.9%)	19.00 (+518.4%)
Amazon (Photo)	7487	Obj.	**2733**	**2733**	725	2705 (+273.1%)	2708 (+273.5%)	**2712** (**+274.1%**)
		Time	38.80	39.02	66.53	2.00 (−97.0%)	4.91 (−92.6%)	32.96 (−50.5%)
Amazon (Computers)	13 381	Obj.	**4829**	**4829**	1281	4773 (+272.6%)	4782 (+273.3%)	**4783** (**+273.4%**)
		Time	188.61	61.78	235.80	3.02 (−98.7%)	8.63 (−96.3%)	79.19 (−66.4%)
Coauthor (CS)	18 333	Obj.	**7506**	**7506**	6635	7479 (+12.7%)	7479 (+12.7%)	**7483** (**+12.8%**)
		Time	1.50	0.09	197.39	3.03 (−98.5%)	13.23 (−93.3%)	122.44 (−38.0%)
Coauthor (Physics)	34 493	Obj.	11 351	**11 353**	2156	11 176 (+418.4%)	11 186 (+418.8%)	**11 190** (**+419.0%**)
		Time	1802.80	81.34	1564.13	8.63 (−99.5%)	38.71 (−97.5%)	385.93 (−75.3%)
SNAP (web-Stanford)	281 903	Obj.	163 385	**163 391**	OB	160 784	160 837	**160 872**
		Time	996.25	44.85		13.35	137.98	1357.11
SNAP (web-NotreDame)	325 729	Obj.	251 846	**251 850**	OB	250 365	250 384	**250 409**
		Time	64.97	1802.34		11.81	119.48	1046.48
SNAP (web-BerkStan)	685 230	Obj.	125 194	**408 483**	OB	403 166	403 189	**403 231**
		Time	1875.96	100.17		88.67	975.35	9940.93
SNAP (soc-Pokec)	1 632 803	Obj.	OB	**788 907**	OB	784 843	**784 891**	OB
		Time		1805.95		770.55	7512.06	
SNAP (wiki-topcats)	1 791 489	Obj.	OB	**986 180**	OB	958 980	**959 051**	OB
		Time		1060.09		850.80	9121.85	

are still remarkable, e.g., the model trained on ER-$(40, 50)$ dataset almost achieves the best score in BA-$(400, 500)$ dataset.

Next, we consider the generalization of the algorithm to large graphs with the number of vertices up to two million. We train ADP and S2V-DQN models on BA-$(400, 500)$ dataset and evaluate them on the following real-world graph datasets: Citation (Sen et al., 2008), Coauthor, Amazon (Shchur et al., 2018) and Stanford Network Analysis Platform (SNAP, Leskovec & Sosič 2016). We additionally evaluate on BA graphs with millions of vertices. Similar to the experiments in Table 2, we set the time limit on CPLEX and ReduMIS by 1800 seconds. The ES method is again excluded as being computationally prohibitive to compare. As reported in Table 4, ADP successfully generates solutions for large scale instances which scale up to two million (2M), even though it was trained on graphs of size smaller than five hundred vertices. Most notably, ADP-10 outperforms the ReduMIS (state-of-the-art solver) in BA graph with 2M vertices, but six times faster. It also outperforms the CPLEX in the graphs with more than 0.5M vertices, indicating better scalability of our algorithm. Note that CPLEX also fails to generate solutions on graphs with more than 1M vertices. Such a result strongly supports the potential of ADP being a generic solver that could be used in place of conventional solvers. On the contrary, we found that S2V-DQN does not generalize well to large graphs: it performs worse and takes much more time for generation of solution as it requires number of decisions proportional to the graph size.

4 Conclusion

In this paper, we propose a new reinforcement learning framework for combinatorial problems that is scalable to large graphs. Our main contribution is the auto-deferring policy, which allows the agent to defer the determinations on vertices for efficient expression of complex structures in the solutions. Through extensive experiments, our algorithm shows performance that is both superior to the existing reinforcement learning baseline and competitive with the highly optimized conventional solvers. We expect our framework can be influential to future works that require policies with actions of high dimension and complex structure.

REFERENCES

Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of satisfiability*, volume 185. IOS press, 2009.

Ravi Boppana and Magnús M Halldórsson. Approximating maximum independent sets by excluding subgraphs. *BIT Numerical Mathematics*, 32(2):180–196, 1992.

Alfredo Braunstein, Marc Mézard, and Riccardo Zecchina. Survey propagation: An algorithm for satisfiability. *Random Structures & Algorithms*, 27(2):201–226, 2005.

Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. Learning heuristics for the tsp by policy gradient. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pp. 170–181. Springer, 2018.

Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

Thomas A Feo, Mauricio GC Resende, and Stuart H Smith. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42(5):860–878, 1994.

Eleanor J Gardiner, Peter Willett, and Peter J Artymiuk. Graph-theoretic techniques for macromolecular docking. *Journal of Chemical Information and Computer Sciences*, 40(2):273–279, 2000.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.

Frank Harary and Ian C Ross. A procedure for clique detection using the group matrix. *Sociometry*, 20(3):205–215, 1957.

Johan Hastad. Clique is hard to approximate within n/sup 1-/spl epsiv. In *Proceedings of 37th Conference on Foundations of Computer Science*, pp. 627–636. IEEE, 1996.

Holger H Hoos and Thomas Stützle. Satlib: An online resource for research on sat. *Sat*, 2000: 283–292, 2000.

Cplex optimization studio. ILOG, IBM, 2014. URL http://www.ibm.com/software/commerce/optimization/cplex-optimizer.

Libin Jiang and Jean Walrand. A distributed csma algorithm for throughput and utility maximization in wireless networks. *IEEE/ACM Transactions on Networking (ToN)*, 18(3):960–972, 2010.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pp. 6348–6358, 2017.

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

Donald Ervin Knuth. *The art of computer programming*, volume 3. Pearson Education, 1997.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ByxBFsRqYm.

Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F Werneck. Finding near-optimal independent sets at scale. *Journal of Heuristics*, 23(4):207–229, 2017.

Jure Leskovec and Rok Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1, 2016.

Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems*, pp. 539–548, 2018.

Marc Mezard, Marc Mezard, and Andrea Montanari. *Information, physics, and computation*. Oxford University Press, 2009.

Raymond E Miller and David E Muller. A problem of maximum consistent subsets. Technical report, IBM Research Report RC-240, JT Watson Research Center, Yorktown Heights, NY, 1960.

Pedro V Sander, Diego Nehab, Eden Chlamtac, and Hugues Hoppe. Efficient traversal of mesh edges using adjacency primitives. *ACM Transactions on Graphics (TOG)*, 27(5):144, 2008.

Thomas J Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pp. 216–226. ACM, 1978.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pp. 2692–2700, 2015.

Bernhard Friedrich Voigt. Der handlungsreisende, wie er sein soll und was er zu thun hat, um aufträge zu erhalten und eines glücklichen erfolgs in seinen geschäften gewiss zu sein. *Commis-Voageur, Ilmenau*, 1831.

Ronald J Williams and Jing Peng. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268, 1991.

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

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1365–1374. ACM, 2015.

A Implementation Details

In this section, we provide additional details for our implementation of the experiments.

Normalization of feature and reward. The iteration-index of MDP used for input of the policy and value networks was normalized by the maximum number of iterations. Furthermore both the MIS objective and the solution diversification rewards were normalized by maximum number of vertices in the corresponding dataset.

Hardware. Computations for our method and S2V-DQN were done on a NVIDIA RTX 2080Ti GPU and a NVIDIA TITAN X Pascal GPU, respectively. Experiments for ES, CPLEX and ReduMIS were run in AWS EC2 c5 instances with Intel Xeon Platinum 8124M CPU. We additionally let the CPLEX use 16 CPU cores as it allows multi-processing.

Hyperparameter. Every hyperparameter was optimized on a per graph type basis and used across all sizes within each graph type. Throughout every experiment, policy and value networks were parameterized by graph convolutional network with 4 layers and 128 hidden dimensions. Every instance of the model was trained for 20000 updates of proximal policy optimization (Schulman et al., 2017), based on the Adam optimizer with learning rate of 0.0001. Validation dataset was used for choosing the best performing model while using 10 samples for evaluating the performance. Reward was not decayed throughout the episodes of the Markov decision process. Gradient norms were clipped by the value of 0.5. We further provide details specific to each type of datasets in Table 5.

Table 5: Choice of hyperparameters for the experiments on performance evaluation. The REDDIT column indicates hyperparameters used for the REDDIT (BINARY, MULTI-5K, MULTI-12K) datasets.

Parameters	ER	BA	SATLIB	PPI	REDDIT	as-Caida
Maximum iterations per episode	32	32	128	128	64	128
Number of unrolling iteration	32	32	128	128	64	128
Number of environments (graph instances)	32	32	32	10	64	1
Batch size for gradient step	16	16	8	8	16	8
Number of gradient steps per update	4	4	8	8	16	8
Solution diversity reward coefficient	0.1	0.1	0.01	0.1	0.1	0.1
Maximum entropy coefficient	0.1	0.1	0.01	0.001	0.0	0.1

Baselines. We implemented the S2V-DQN algorithm based on the code (written in C++) provided by the authors.[3] Note that the provided code solves the minimum vertex cover (MVC) problem, which is equivalent to solving the MIS problem. Specifically, complement of every vertex cover is an independent set. Then complement of minimum vertex cover is a maximum independent set, which implies two problems are equivalent. For ER and BA models, S2V-DQN was unstable to be trained on graphs of size from $(100, 200)$ and $(400, 500)$ without pre-training. Instead we performed fine-tuning as mentioned in the original paper (Khalil et al., 2017). First, for the ER-$(100, 200)$ and BA-$(100, 200)$ datasets, we fine-tuned the model trained on ER-$(50, 100)$ and BA-$(50, 100)$, respectively. Next, for the ER-$(400, 500)$ and BA-$(400, 500)$ datasets, we performed "curriculum learning", e.g., a model was first trained on the ER-$(50, 100)$ dataset, then fine-tuned on the ER-$(100, 200)$, ER-$(200, 300)$, ER-$(300, 400)$ and ER-$(400, 500)$ in sequence. Finally, for training S2V-DQN on large graphs used in Table 2, we were unable to train on the raw graph under available computational budget. Hence we trained S2V-DQN on subgraphs sampled from the training graphs. To this end, we sampled edges from the model uniformly at random without replacment, until the number of vertices reach 300. Then we used the subgraph induced from the sampled vertices. For other algorithms, we run ES algorithm and ReduMIS based on NetworkX package[4] and code provided by the authors,[5] respectively.

[3]https://github.com/Hanjun-Dai/graph_comb_opt
[4]https://networkx.github.io/
[5]http://algo2.iti.kit.edu/kamis/

Table 6: Number of nodes, edges and graphs for each dataset used in the Table 2. Number of graphs indicates the number of training, validation and test graphs, respectively.

Dataset	Number of nodes	Number of edges	Number of graphs
SATLIB	(1209, 1347)	(4696, 6065)	(38 000, 1000, 1000)
PPI	(591, 3480)	(3854, 53 377)	(20, 2, 2)
REDDIT (BINARY)	(6, 3782)	(4, 4071)	(1600, 200)
REDDIT (MULTI-5K)	(22, 3648)	(21, 4783)	(4001, 499, 499)
REDDIT (MULTI-12K)	(2, 3782)	(1, 5171)	(9545, 1192, 1192)
as-Caida	(8020, 26 475)	(36 406, 106 762)	(108, 12, 12)

Table 7: Number of nodes and edges for each dataset used in the Table 4.

Dataset	Number of nodes	Number of edges
Citeseer	3327	3668
Cora	2708	5069
Pubmed	19 717	44 324
Coauthor CS	18 333	81 894
Coauthor Physics	34 493	247 962
Amazon Computers	13 381	245 778
Amazon Photo	7487	119 043
web-Stanford	281 903	2 312 497
web-NotreDame	325 729	1 497 134
web-BerkStan	685 230	7 600 595
soc-Pokec	1 632 803	30 622 564
wiki-topcats	1 791 489	28 511 807
BA-1M	1 000 000	3 999 984
BA-2M	2 000 000	7 999 984

B Dataset Details

In this section, we provide additional details on the datasets used for the experiments.

ER and BA datasets. For the ER and BA datasets, we train on graphs randomly generated on the fly, and perform validation and evaluation on a fixed set of 1000 graphs.

SATLIB dataset. The SATLIB dataset is a popular benchmark for evaluating SAT algorithms. We specifically use the synthetic problem instances from the category of random 3-SAT instances with controlled backbone size (Singer et al., 2000). Next, we describe the procedure for reducing the SAT instances to MIS instances. To this end, a vertex is added to the graph for each literal of the SAT instance. Then edges are added for each pair of vertices satisfying the following conditions: (a) that are in the same clause or (b) they correspond to the same literals with different signs. Consequently, the MIS in the resulting graph correspond to the truth assignment to the optimal assignments of the SAT problem (Dasgupta et al., 2008).

PPI dataset. The PPI dataset is the protein-protein-interaction dataset with vertices representing proteins and edges representing interactions between them.

REDDIT datasets. The REDDIT (BINARY, MULTI-5K, MULTI-12K) datasets are constructed from online discussion threads in reddit[6] where vertices represent to users and edge exists between two vertices if at one of them responded to anothers comment.

Autonomous system dataset. The as-Caida dataset is set of autonomous system graphs derived from a set of RouteViews BGP table snapshots (Leskovec et al., 2005).

Citation dataset. The Cora and the Citeseer are networks constructed by vertices and edges representing documentation and citation links between them, respectively (Sen et al., 2008).

Amazon dataset. The Computers the Photo graphs are segmented from the Amazon co-purchase graph (McAuley et al., 2015), where vertices corresponds to goods and edges between pair of vertices if they are frequently purchased together.

Coauthor dataset. The CS and Physics graphs represents authors and the corresponding co-authorships by vertices and edges, respectively. It was collected from Microsoft Academic Graph from the KDD Cup 2016 challenge3 .[7]

Web network datasets. The web-NotreDame, web-Stanford and web-BerkStan are graphs with vertices represent web-pages obtained from the University of Notre Dame, Stanford University and Berkeley & Stanford University domains, respectively (Leskovec & Sosič, 2016). Edge is added for each pair of web-pages with links between them.

Social network dataset. The soc-Pokec is a graph representing friendships between users from a social network in Slovakia (Takac & Zabovsky, 2012).

Wikipedia network dataset. The wiki-topcats graph is constructed by collecting connected pages which belong to top categories containing at least 100 pages, starting from the most connected component of Wikipedia (Klymko et al., 2014).

We further provide the statistics of the dataset used in experiments corresponding to Table 2 and 4 in Table 6 and 7, respectively.

References

Sanjoy Dasgupta, Christos H Papadimitriou, and Umesh Virkumar Vazirani. *Algorithms*. McGraw-Hill Higher Education, 2008.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pp. 6348–6358, 2017.

Christine Klymko, David Gleich, and Tamara G Kolda. Using triangles to improve community detection in directed networks. *arXiv preprint arXiv:1404.5874*, 2014.

Jure Leskovec and Rok Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1, 2016.

Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pp. 177–187. ACM, 2005.

Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 43–52. ACM, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[6] `https://www.reddit.com/`
[7] `https://kddcup2016.azurewebsites.net/`

Josh Singer, Ian P Gent, and Alan Smaill. Backbone fragility and the local search cost peak. *Journal of Artificial Intelligence Research*, 12:235–270, 2000.

Lubos Takac and Michal Zabovsky. Data analysis in public social networks. In *International Scientific Conference and International Workshop Present Day Trends of Innovations*, volume 1, 2012.