# SOLVING NP-HARD PROBLEMS ON GRAPHS WITH EXTENDED ALPHAGO ZERO

A PREPRINT

**Kenshin Abe** [1] [*], **Zijian Xu** [1] [*], **Issei Sato** [1, 2], **Masashi Sugiyama** [2, 1]
[1] The University of Tokyo, [2] RIKEN
abe@ms.k.u-tokyo.ac.jp, xuzijian@ms.k.u-tokyo.ac.jp,
sato@k.u-tokyo.ac.jp, sugi@k.u-tokyo.ac.jp

## ABSTRACT

There have been increasing challenges to solve combinatorial optimization problems by machine learning. Khalil et al. [18] proposed an end-to-end reinforcement learning framework, S2V-DQN, which automatically learns graph embeddings to construct solutions to a wide range of problems. To improve the generalization ability of their Q-learning method, we propose a novel learning strategy based on AlphaGo Zero [34] which is a Go engine that achieved a superhuman level without the domain knowledge of the game. Our framework is redesigned for combinatorial problems, where the final reward might take any real number instead of a binary response, win/lose. In experiments conducted for five kinds of NP-hard problems including MINIMUMVERTEXCOVER and MAXCUT, our method is shown to generalize better to various graphs than S2V-DQN. Furthermore, our method can be combined with recently-developed graph neural network (GNN) models such as the *Graph Isomorphism Network* [38], resulting in even better performance. This experiment also gives an interesting insight into a suitable choice of GNN models for each task.

## 1 Introduction

There is no polynomial-time algorithm found for NP-hard problems [7], but they often arise in many real-world optimization tasks. Therefore, a variety of algorithms have been developed, including approximation algorithms [2, 13], meta-heuristics based on local searches such as simulated annealing and evolutionary computation [10, 14], general-purpose exact solvers such as CPLEX [2] and Gurobi [15], and problem-specific exact solvers. Some problem-specific exact solvers are fast and practical for certain well-studied combinatorial problems such as MINIMUMVERTEXCOVER [1, 24]. These algorithms can handle sparse graphs of millions of nodes. However, they usually require problem-specific highly sophisticated reduction rules and heavy implementation and are hard to generalize for other combinatorial problems. Moreover, real-world NP-hard problems are usually much more complex than these problems and thus it is difficult to construct ad-hoc efficient algorithms for them.

Recently, machine learning approaches have been actively investigated to solve combinatorial optimization, with the expectation that the combinatorial structure of the problem can be automatically learned without complicated hand-crafted heuristics. In the early stage, many of these approaches focused on solving specific problems [5, 16] such as the traveling salesperson problem (TSP). Recently, Khalil et al. [18] proposed a framework to solve combinatorial problems by a combination of reinforcement learning and graph embedding, which attracted attention for the following two reasons: It does not require any knowledge on graph algorithms other than greedy selection based on network outputs. Furthermore, it learns algorithms without any training dataset. Thanks to these advantages, the framework can be applied to a diverse range of problems over graphs and it also performs much better than previous learning-based approaches. However, we observed poor empirical performance on some graphs having different characteristics (e.g., synthetic graphs and real-world graphs) than random graphs that were used for training, possibly because of the limited exploration space of their Q-learning method.

[*] equal contribution
[2] www.cplex.com

In this paper, to overcome this weakness, we propose to replace Q-learning with our novel learning strategy named CombOpt Zero. CombOpt Zero is inspired by AlphaGo Zero [32], a superhuman engine of Go, which conducts Monte Carlo Tree Search (MCTS) to train deep neural networks. AlphaGo Zero was later generalized to AlphaZero [33] so that it can handle other games; however, its range of applications is limited to two-player games whose state is win/lose (or possibly draw). We extend AlphaGo Zero to a bunch of combinatorial problems by a simple normalization technique based on random sampling and show that it successfully works in experiments. More specifically, we train our networks for five kinds of NP-hard tasks and test on different instances including standard random graphs (e.g., the Erdős-Renyi model [11] and the Barabási-Albert model [3]) and many real-world graphs. We show that CombOpt Zero has a better generalization to a variety of graphs than the existing method, which indicates the MCTS based training strengthens the exploration of various actions. Also, using the MCTS at test time significantly improves performance for a certain problem and guarantees the solution quality. Furthermore, we propose to combine our framework with several Graph Neural Network models [20, 27, 38], and experimentally demonstrate that an appropriate choice of models contributes to improving the performance with a significant margin.

## 2 Preliminary

In this section, we introduce the main ingredients which our work is based on.

### 2.1 Notations

In this paper, we use $G = (V, E)$ to denote an undirected and unlabelled graph, where $V$ is the set of vertices and $E$ is the set of edges. Since $v$ is used to represent a state value in AlphaGo Zero, we often use $x, y, ...$ to denote a node of a graph. $V(G)$ indicates the set of vertices of graph $G$. $\mathcal{N}(x)$ means the set of 1-hop neighbors of node $x$ and $\mathcal{N}(S) = \bigcup_{x \in S} \mathcal{N}(x)$. We usually use $a, b, \ldots$ for actions. Bold variables such as $\boldsymbol{p}$ and $\boldsymbol{\pi}$ are used for vectors.

### 2.2 Machine Learning for Combinatorial Optimization

Machine learning approaches for combinatorial optimization problems have been studied in the literature, starting from Hopfield and Tank [16], who applied a variant of neural networks to small instances of Traveling Salesperson Problem (TSP). With the success of deep learning, more and more studies were conducted including Bello et al. [5], Kool et al. [22] for TSP and Wang et al. [36] for MAXSAT.

Khalil et al. [18] proposed an end-to-end reinforcement learning framework S2V-DQN, which attracted attention because of promising results in a wide range of problems over graphs such as MINIMUMVERTEXCOVER and MAXCUT. It optimizes a deep Q-network (DQN) where the Q-function is approximated by a graph embedding network, called `structure2vec` (S2V) [9]. The DQN is based on their reinforcement learning formulation, where each action is picking up a node and each state represents the "sequence of actions". In each step, a partial solution $S \subset V$, i.e., the current state, is expanded by the selected vertex $v^* = \underset{v \in V(h(S))}{\operatorname{argmax}} Q(h(S), v)$ to $(S, v^*)$, where $h(\cdot)$ is a fixed function determined by the problem that maps a state to a certain graph, so that the selection of $v$ will not violate the problem constraint. For example, in MAXIMUMINDEPENDENTSET, $h(S)$ corresponds to the induced subgraph of the input graph $G = (V, E)$, where vertices are limited to $V \backslash (S \cup \mathcal{N}(S))$. The immediate reward is the change in the objective function. The Q-network, i.e, S2V learns a fixed dimensional embedding for each node.

In this work, we mitigate the issue of S2V-DQN's generalization ability. We follow the idea of their reinforcement learning setting, with a different formulation, and replace their Q-learning by a novel learning strategy inspired by AlphaGo Zero. Note that although some studies combine classic heuristic algorithms and learning-based approaches (using dataset) to achieve the state-of-the-art performance [25], we stick to learning without domain knowledge and dataset in the same way as S2V-DQN.

### 2.3 AlphaGo Zero

AlphaGo Zero [34] is a well-known superhuman engine designed for use with the game of Go. It trains a deep neural network $f_\theta$ with parameter $\theta$ by reinforcement learning. Given a state (game board), the network outputs $f_\theta(s) = (\boldsymbol{p}, v)$, where $\boldsymbol{p}$ is the probability vector of each move and $v \in [-1, 1]$ is a scalar denoting the value of the state. If $v$ is close to 1, the player who takes a corresponding action from state $s$ is very likely to win.

The fundamental idea of AlphaGo Zero is to enhance its own networks by self-play. For this self-play, a special version of Monte Carlo Tree Search (MCTS) [21], which we describe later, is used. The network is trained in such a way that the policy imitates the enhanced policy by MCTS $\boldsymbol{\pi}$, and the value imitates the actual reward from self play $z$ (i.e. $z = 1$

if the player wins and $z = -1$ otherwise). More formally, it learns to minimize the loss

$$\mathcal{L} = (z - v)^2 + \mathrm{CrossEntropy}(\boldsymbol{p}, \boldsymbol{\pi}) + c_{\mathrm{reg}}\|\theta\|_2^2, \tag{1}$$

where $c_{\mathrm{reg}}$ is a nonnegative constant for $L_2$ regularization.

MCTS is a heuristic search for huge tree-structured data. In AlphaGo Zero, the search tree is a rooted tree, where each node corresponds to a state and the root is the initial state. Each edge $(s, a)$ denotes action $a$ at state $s$ and stores a tuple

$$(N(s, a), W(s, a), Q(s, a), P(s, a)), \tag{2}$$

where $N(s, a)$ is the visit count, $W(s, a)$ and $Q(s, a)$ are the total and mean action value respectively, and $P(s, a)$ is the prior probability. One iteration of MCTS consists of three parts: *select*, *expand*, and *backup*. First, from the root node, we keep choosing an action that maximizes an upper confidence bound

$$Q(s, a) + c_{\mathrm{puct}}P(s, a)\frac{\sqrt{\sum_{a'} N(s, a')}}{1 + N(s, a)}, \tag{3}$$

where $c_{\mathrm{puct}}$ is a non-negative constant (*select*). Once it reaches to unexplored node $s$, then the edge values (2) are initialized using the network prediction $(\boldsymbol{p}, v) = f_\theta(s)$ (*expand*). After expanding a new node, each visited edge is traversed and its edge values are updated (*backup*) so that $Q$ maintain the mean of state evaluations over simulations:

$$Q(s, a) = \frac{1}{N(s, a)} \sum_{s' | s, a \to s'} v_{s'}, \tag{4}$$

where the sum is taken over those states reached from $s$ after taking action $a$. After some iterations, the probability vector $\pi$ is calculated by

$$\boldsymbol{\pi}_a = \frac{N(s_0, a)^{1/\tau}}{\sum_b N(s_0, b)^{1/\tau}} \tag{5}$$

for each $a \in A_{s_0}$, where $\tau$ is a temperature parameter.

AlphaGo Zero defeated its previous engine AlphaGo [32] with 100-0 score without human knowledge, i.e., the records of the games of professional players and some known techniques in the history of Go. We are motivated to take advantage of the AlphaGo Zero technique in our problem setting since we also aim at training deep neural networks for discrete optimization without domain knowledge. However, we cannot directly apply AlphaGo Zero, which was designed for two-player games, to combinatorial optimization problems. Section 3.2 and 3.3 explains how we manage to resolve this issue.

## 2.4 Graph Neural Network

A Graph Neural Network (GNN) is a neural network that takes graphs as input. Kipf and Welling [20] proposed the *Graph Convolutional Network* (GCN) inspired by spectral graph convolutions. Because of its scalability, many variants of spatial based GNN were proposed. Many of them can be described as a Message Passing Neural Network (MPNN) [12]. They recursively aggregate neighboring feature vectors to obtain node embeddings that capture the structural information of the input graph. The *Graph Isomorphism Network* (GIN) [38] is one of the most expressive MPNNs in terms of graph isomorphism. Although they have a good empirical performance, some studies point out the limitation of the representation power of MPNNs [29, 38].

Maron et al. [26] proposed an Invariant Graph Network (IGN) using tensor representations of a graph and was shown to be universal [17, 28]. Since it requires a high-order tensor in middle layers, which is impractical, Maron et al. [27] proposed *2-IGN+*, a scalable and powerful model.

All of these models, as well as S2V [9] used in S2V-DQN, are compared in the experiments to test the difference in the performance for combinatorial optimization. The detail of each model is described in Appendix B.

## 2.5 NP-hard Problems on Graphs

Here, we introduce three problems out of the five NP-hard problems we conducted experiments on. The other two problems are described in Appendix A.

**Minimum Vertex Cover** A subset of nodes $V' \subset V$ is called a vertex cover if all edges are covered by $V'$; for all $(x, y) \in E$, $x \in V'$ or $y \in V'$ holds. MINIMUMVERTEXCOVER asks a vertex cover whose size is minimum.

**Max Cut** Let $C \subset E$ be a cut set of between $V' \subset V$ and $V \backslash V'$, i.e., $C = \{(u, v) \in E \mid u \in V', v \in V \backslash V'\}$. MAXCUT asks for a subset $V'$ that maximizes the size of cut set $C$.

**Maximum Clique** A subset of nodes $V' \subset V$ is called a clique if any two nodes in $V'$ are adjacent in the original graph; for all $x, y \in V'$ $(x \neq y)$, $(x, y) \in E$. MAXIMUMCLIQUE asks for a largest clique.

Note that for all problems in the experiments, we are focusing on the unweighted case. In Section 3.1, we show that all of these problems can be formulated in our framework.

## 3 Method

In this section, we give a detailed explanation of our algorithm to solve combinatorial optimization problems over graphs. First, we introduce a reinforcement learning formulation of NP-hard problems over graphs and show that the problems introduced in Section 2.5 can be accommodated under this formulation. Next, we explain the basic ideas of CombOpt Zero in light of the difference between 2-player games and combinatorial optimization. Finally, we propose the whole algorithm along with pseudocodes.

### 3.1 Reduction to MDP

To apply the AlphaGo Zero method to combinatorial optimization, we reduce graph problems into a Markov Decision Process (MDP) [4]. A deterministic MDP is defined as $(S, A_s, T, R)$, where $S$ is a set of states, $A_s$ is a set of actions from the state $s \in S$, $T : S \times A_s \to S$ is a deterministic state transition function, and $R : S \times A_s \to \mathbb{R}$ is an immediate reward function. In our problem setting on graphs, each state $s \in S$ is represented as a *labeled* graph, a tuple $s = (G, d)$. $G = (V, E)$ is a graph and $d : V \to L$ is a node-labeling function, where $L$ is a label space.

For each problem, we have a set of terminal states $S_{\text{end}}$. Given a state $s$, we repeat selecting an action $a$ from $A_s$ and transiting to the next state $T(s, a)$, until $s \in S_{\text{end}}$ holds. By this process, we get a sequence of states and actions $[s_0, a_0, s_1, a_1, ..., a_{N-1}, s_N]$ of $N$ steps where $s_N \in S_{\text{end}}$, which we call a trajectory. For this trajectory, we can calculate the simple sum of immediate rewards $\sum_{n=0}^{N-1} R(s_n, a_n)$, and we define $r^*(s)$ be the maximum possible sum of immediate rewards out of all trajectories from state $s$. Let Init be a function that maps the input graph to the initial state. Our goal is to, given graph $G_0$, obtain the maximum sum of rewards $r^*(\text{Init}(G_0))$.

Below, we show that the problems described in Section 2.5 can be naturally accommodated in this framework, by defining $L$, $A_s$, $T$, $R$, Init, and $S_{\text{end}}$ as follows:

**Minimum Vertex Cover** Since we do not need a label of the graph for this problem, we set $d$ to a constant function on any set $L$ (e.g., $L = \mathbb{R}$, $d(s) = 1$). Actions are represented by selecting one node ($A_s = V$). Init uses the same graph as $G_0$ and $d$ defined above. $T(s, x)$ returns the next state, corresponding to the graph where edges covered by $x$ and isolated nodes are deleted. $S_{\text{end}}$ is the states with the empty graphs. $R(s, x) = -1$ for all $s$ and $x$ because we want to minimize the number of transition steps in the MDP.

**Max Cut** In each action, we color a node by 0 or 1, and remove it while each node keeps track of how many adjacent nodes have been colored with each color. $A_s = \{(x, c) \mid x \in V, c \in \{1, 2\}\}$ denotes a set of possible coloring of a node, where $(x, c)$ means coloring node $x$ with color $c$. $L = \mathbb{N}^2$, representing the number of colored (and removed) nodes in each color (i.e., $l_0$ is the number of (previously) adjacent nodes of $x$ colored with 0, and same for $l_1$, where $l = d(x)$). Init uses the same graph as $G_0$ and sets both of $d(x)$ be 0 for all $x \in V$. $T(s, (x, c))$ increases the $c$-th value of $d(x')$ by one for $x' \in N(x)$ and removes $x$ and neighboring edges from the graph. $S_{\text{end}}$ is the states with the empty graphs. $R(s, (x, c))$ is the $(3 - c)$-th (i.e., 2 if $c = 1$ and 1 if $c = 2$) value of $d(x)$, meaning the number of edges in the original graph which have turned out to be included in the cut set (i.e., colors of the two nodes are different).

**Maximum Clique** $d$, $A_s$, Init, and $S_{\text{end}}$ are the same as MINIMUMVERTEXCOVER. $T(s, x)$ return the next state whose corresponding graphs is the induced subgraph of $\mathcal{N}(x)$; 1-hop neighbors of $x$. $R(s, x) = 1$ because we want to maximize the number of transition steps of the MDP.

Figure 1 shows some possible MDP transitions for MAXCUT. It is easy to check that, for all reduction rules described above, finding a sequence that maximizes the sum of rewards is equivalent to solving the original problems. One of the differences from the formulation of Khalil et al. [18] is that we do not limit the action space to a set of nodes. This enables more flexibility and results in different reduction rules in some problems. For example, in our formulation of MAXCUT, actions represent *a node coloring*. See Appendix E for more examples of the differences.

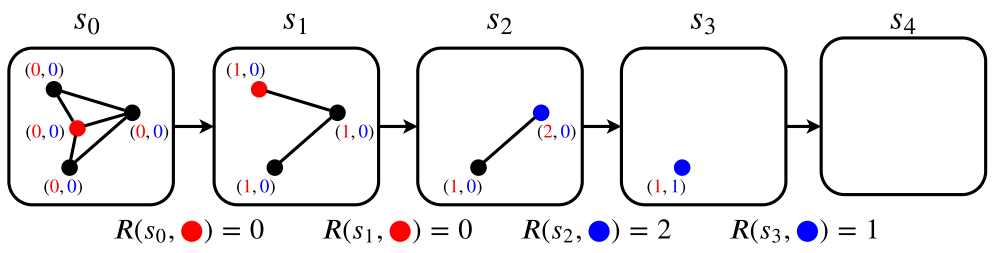

Figure 1: **Example of MAXCUT MDP trainsitions.** In the transition sequence, the upper left and center node are colored in ●, the right and lower left node are colored in ●, resulting in the cumulative reward 3.

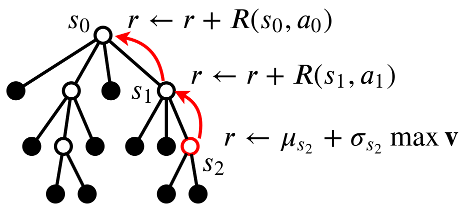
$s_0$
$r \leftarrow r + R(s_0, a_0)$
$s_1$
$r \leftarrow r + R(s_1, a_1)$
$r \leftarrow \mu_{s_2} + \sigma_{s_2} \max \mathbf{v}$
$s_2$

Figure 2: **Backup.** After estimating the reward of the expanded node, we iteratively update the rewards of its ancestors.

## 3.2 Ideas for CombOpt Zero

Here, we explain the basic idea behind our extension of AlphaGo Zero into combinatorial optimization problems.

**Combinatorial Optimization vs. Go** When we try to apply the AlphaGo Zero method to our MDP formulation of combinatorial optimization, we need to consider the following two differences from Go. First, in our problem setting, the states are represented by (labeled) graphs of different sizes, while the boards of Go can be represented by $19 \times 19$ fixed-size matrix. This can be addressed easily by adopting GNN models instead of convolutional neural networks in AlphaGo Zero, just like S2V-DQN. The whole state $s = (G, d)$ can be given to GNN, where the coloring $d$ is used as the node feature. The second difference is the possible answers for a state: While the final result of Go is only win or lose, the answer to combinatorial problems can take any integers or even real numbers. This makes it easy to model the state value of Go by "how likely the player is going to win", with the range of $[-1, 1]$, meaning that the larger the value is, the more likely the player is to win. Imitating this, in our problem setting, one can evaluate the state value in such a way that it predicts the maximum possible sum of rewards from the current state. However, since the answer might grow infinitely usually depending on the graph size, this naive approximation strategy makes it difficult to balance the scale in (1) and (3). For example, in (3), the first term could be too large when the answer is large, causing less focus on $P(s, a)$ and $N(s, a)$. To mitigate this problem, we propose a reward normalization technique.

Given a state $s$, the network outputs $(\boldsymbol{p}, \boldsymbol{v}) = f_\theta(s)$. While $\boldsymbol{p}$ is the same as AlphaGo Zero (action probabilities), $\boldsymbol{v}$ is a *vector* instead of a scalar value. $v_a$ predicts the *normalized reward* of taking action $a$ from state $s$, meaning "how good the reward is compared to the return obtained by random actions". Formally, we train the network so that $\boldsymbol{v_a}$ predicts $(R(s, a) + r^*(T(s, a)) - \mu_s)/\sigma_s$, where $\mu_s$ and $\sigma_s$ are the mean and the standard deviation of the cumulative rewards by random plays from state $s$. When we estimate the state value of $s$, we consider taking action $a$ which maximizes $\boldsymbol{v}_a$ and restoring the unnormalized value by $\mu_s$ and $\sigma_s$:

$$r_{\text{estim}}(s) = \begin{cases} 0 & (s \in S_{\text{end}}) \\ \mu_s + \sigma_s \cdot (\max_{a \in A_s} \boldsymbol{v}_a) & (otherwise). \end{cases} \tag{6}$$

Similarly, we also let $W(s, a)$ and $Q(s, a)$ hold the sum and mean of *normalized* action value estimations over the MCTS iterations. Figure 2 illustrates the idea of how we update these values efficiently in the *backup* phase. See Section 3.3 for the details.

**Reward Normalization Technique** By virtue of this normalization, we no longer need to care about the difference of scales in (1) and (3). Moreover, it frees the algorithm from problem specification, that is, when we are to maximize some criterion, we always evaluate the action by "how good it is compared to random actions".

**Algorithm 1** MCTS

```
Require: Network $f_\theta$, root node $s_0$
Ensure: Return $\boldsymbol{\pi}$, enhanced policy
  while $\sum_{a \in A_{s_0}} N(s_0, a) \le c_{\text{iter}}|A_{s_0}|$ do
    $s = s_0$
    {select}
    while $s$ is expanded before and $s \notin S_{\text{end}}$ do
      $a = \underset{b \in A_s}{\text{argmax}}\left(Q(s,b) + c_{\text{puct}} P(s,b) \frac{\sqrt{\sum_{b'} N(s,b')}}{1+N(s,b)}\right)$
      $s \leftarrow T(s,a)$
    end while
    {expand}
    if $s \notin S_{\text{end}}$ then
      $(\boldsymbol{p}, \boldsymbol{v}) = f_\theta(s)$
      initialize $(N(s,a) = 0, W(s,a) = 0, Q(s,a) = 0, P(s,a) = \boldsymbol{p_a})$ for each $a \in A_s$
      Calculate $(\mu_s, \sigma_s)$ by random sampling
    end if
    {backup}
    $r = r_{\text{estim}}(s)$
    while $s$ is not $s_0$ do
      $a =$ previous action
      $s \leftarrow$ parent of $s$
      $r \leftarrow r + R(s,a)$
      $r' = (r - \mu_s)/\sigma_s$
      $W(s,a) \leftarrow W(s,a) + r'$
      $N(s,a) \leftarrow N(s,a) + 1$
      $Q(s,a) \leftarrow \frac{W(s,a)}{N(s,a)}$
    end while
  end while
  Compute $\left(\boldsymbol{\pi}_a = \frac{N(s_0,a)^{1/\tau}}{\sum_b N(s_0,b)^{1/\tau}}\right)$ for each $a$
  return $\boldsymbol{\pi}$
```

## 3.3 Algorithms

Based on the discussion so far, now we are ready to introduce the whole algorithm.

**MCTS Tree Structure** Same as AlphaGo Zero, each edge stores values in (2). Additionally, each node $s$ stores a tuple $(\mu_s, \sigma_s)$, the mean and the standard deviation of the results by random plays.

**MCTS** The pseudocode is available in Algorithm 1. Given an initial state $s_0$, we repeat the iterations of the three parts: *select*, *expand*, and *backup*. *Select* is same as AlphaGo Zero (keep selecting a child node that maximizes (1)). Once we reach unexpanded node $s$, we *expand* the node. $(\boldsymbol{p}, \boldsymbol{v}) = f_\theta(s)$ is evaluated and each edge value is updated as $(N(s,a) = 0, W(s,a) = 0, Q(s,a) = 0, P(s,a) = \boldsymbol{p}_a)$ for $a \in A_s$. At the same time, $(\mu_s, \sigma_s)$ is calculated by random sampling from $s$. In *backup*, for each node $s'$ and the corresponding action $a'$ in a backward pass, $N(s', a')$ is incremented by one in the same way as AlphaGo Zero. The difference is that $Q(s', a')$ is updated to hold the estimation of the *normalized* mean reward from $s'$. We approximate the non-normalized state value by (6) and calculate the estimated cumulative reward from $s'$ by adding the immediate reward each time we move back to the parent node (Figure 2 illustrates this process). $\boldsymbol{\pi}$ is calculated in the same way as AlphaGo Zero (5) after sufficient iterations. While the number of the iterations is fixed in AlphaGo Zero, in CombOpt Zero, since the size of action space differs by states, we make it proportional to the number of actions: $c_{\text{iter}}|A_s|$. Following AlphaGo Zero, we add Dirichlet noise to the prior probabilities only for the initial state $s_0$ to explore the various initial actions.

**Training** Following AlphaGo Zero, the training of CombOpt Zero is composed of three roles: *data generators*, *learners*, and *model evaluators*. The best model that has performed best so far is shared among all of these three components. The *data generator* repeats generating self-play records for a randomly generated graph input, by the MCTS based on the current best model. The records are the sequence of $(s, a, \boldsymbol{\pi}, z')$, which means action $a$ was taken from $s$ depending on the MCTS-enhanced policy $\boldsymbol{\pi}$, and $z' = (z - \mu_R(s))/\sigma_R(s)$ where $z$ is the cumulative reward from $s$ to the terminal state. The *learner* randomly sample mini-batches from the generator's data and update the

**Algorithm 2** Self-play Data Generation

```
Require: Network $f_\theta$, Initial graph $G_0$
Ensure: Return self-play data records
  {self-play}
  $s = \text{Init}(G_0)$
  records $= \{\}$
  while $s \notin S_{\text{end}}$ do
    $\boldsymbol{\pi} = \texttt{MCTS}(s)$
    $a =$ sampled action according to probability $\boldsymbol{\pi}$
    $s \leftarrow T(s, a)$
    Add $(s, a, \boldsymbol{\pi})$ to records
  end while
  {calculate $z'$}
  $z = 0$
  for all $(s, a, \boldsymbol{\pi})$ in records (reversed order) do
    $z \leftarrow z + R(s, a)$
    $z' = (z - \mu_R(s))/\sigma_R(s)$
    Replace $(s, a, \boldsymbol{\pi})$ with $(s, a, \boldsymbol{\pi}, z')$
  end for
  return records
```

parameter of the best model so that it minimizes the loss

$$\mathcal{L} = (z' - \boldsymbol{v}_a)^2 + \text{CrossEntropy}(\boldsymbol{p}, \boldsymbol{\pi}) + c_{\text{reg}} \|\theta\|_2^2. \tag{7}$$

There are two modification from (1): since $\boldsymbol{v}$ is a vector, $\boldsymbol{v}_a$ is used for the loss; since we aim at learning normalized action value function $\boldsymbol{v}$, we use the normalized cumulative reward $z'$ instead of $z$. The *model evaluator* compares the updated model with the best one and stores the better one. In CombOpt Zero, where we cannot compare the winning rate of two players, the evaluator generates random graph instances each time and compare the mean performance on them.

In CombOpt Zero, the data generator needs to calculate $z'$ for the broader reward function $R$ defined in Section 3.1. This can be achieved in the same way as *backup* in MCTS: after generating the self-play trajectory, it first calculates the cumulative sum of immediate rewards in reverse order and normalizes them with $\mu_s$ and $\sigma_s$. In this way, we can compute the normalized rewards from the trajectory effectively. The pseudocode of the data generator is available in Algorithm 2.

## 4 Experiments

In this section, we show some brief results of our experiments on MINIMUMVERTEXCOVER, MAXCUT, and MAXIMUMCLIQUE. Refer to Appendix D for the full results, including the other two problems and further analyses.

**Competitors** Since we aim at solving combinatorial optimization problems without domain knowledge or training dataset, S2V-DQN is our main competitor. Additionally, for each problem, we prepared some known heuristics or approximation algorithms to compare with. For MINIMUMVERTEXCOVER, we implemented a simple randomized 2-approximation algorithm (2-approx) with a reduction rule that looks for degree-1 nodes. The detailed algorithm is explained in Appendix D.1. For MAXCUT, we used a randomized algorithm by semidefinite programming [13] and two heuristics, non-linear optimization with local search [6] and cross-entropy method with local search [23] from MQLib [10].

**Training and Test** We trained the models for two hours to make the training of both CombOpt Zero and S2V-DQN converge (Only 2-IGN+ was trained for four hours due to its slow inference). Note that CombOpt Zero was trained on 4 GPUs while S2V-DQN uses a single GPU because of its implementation, which we discuss in Section 4.4. For the hyperparameters of the MCTS, we referred to the original AlphaGo Zero and its reimplementation, ELF OpenGo [35]. See Appendix C for the detailed environment and hyperparameters. Since we sometimes observed extremely poor performance for a few models both in S2V-DQN and CombOpt Zero when applied to large graphs, we trained five different models from random parameters and took the best value among them at the test time. Throughout the experiments, all models were trained on randomly genrated Erdős-Renyi (ER) [11] graphs with $80 \leq n \leq 100$ nodes and edge probability $p = 0.15$ except for MAXCUT, where nodes were $40 \leq n \leq 50$ and for MAXIMUMCLIQUE, where edge probability was $p = 0.5$. As the input node feature of MAXCUT, we used a two-dimensional feature vector

Table 1: **Generalization Comparision between CombOpt Zero and S2V-DQN.** COZ and DQN are short for CombOpt Zero and S2V-DQN respectively. Smaller is better for MINIMUMVERTEXCOVER and larger is better for MAXCUT.

| | MVC | | MAXCUT | |
|---|---|---|---|---|
| | COZ | DQN | COZ | DQN |
| er100_15 | 76 | 76 | 494 | **527** |
| er1000_5 | 900 | **898** | 12561 | **14424** |
| er5000_1 | 4484 | **4482** | 63389 | **71909** |
| ba100_5 | 63 | 63 | 337 | **343** |
| ba1000_5 | **592** | 594 | **3492** | 3465 |
| ba5000_5 | **2920** | 2927 | **17381** | 16870 |
| ws100_k2_p10 | 49 | 49 | **98** | 97 |
| ws100_k10_p10 | 78 | 78 | 335 | 335 |
| ws1000_k2_p10 | **492** | 496 | **999** | 973 |
| ws1000_k4_p10 | **635** | 636 | **1536** | 1327 |
| ws1000_k10_p10 | 787 | **784** | **3312** | 3287 |
| regular_100_d5 | **63** | 64 | **207** | 205 |
| regular_1000_d5 | **632** | 634 | **2051** | 2046 |
| tree100 | 44 | 44 | **99** | 98 |
| tree1000 | **439** | 440 | **999** | 982 |
| cora | 1258 | 1258 | **4260** | 4243 |
| citeseer | 1462 | **1461** | **3933** | 3893 |
| web-edu | 1451 | 1451 | **4712** | 4289 |
| web-spam | **2299** | 2319 | 20645 | **21027** |
| road-minnesota | **1324** | 1329 | **3080** | 3015 |
| bio-yeast | **456** | 457 | **1769** | 1751 |
| bio-SC-LC | **1039** | 1053 | 10893 | **11890** |
| rt_damascus | 369 | 369 | **3698** | 3667 |
| soc-wiki-vote | 407 | **406** | **2119** | 2064 |
| socfb-bowdoin47 | 1796 | **1793** | **42063** | 37140 |

that stores the number of adjacent nodes of color 1 and color 2. For the other problems, we used a vector of ones. In tests, to keep the fairness, CombOpt Zero conducted a greedy selection on network policy output $\boldsymbol{p}$, which is the same way as S2V-DQN works (except for Section 4.3 where we compared the greedy selection and MCTS).

**Dataset** We generated ER and Barabási-Albert (BA) graphs [3] of different sizes for testing. ER100_15 denotes an ER graph with 100 nodes and edge probability $p = 0.15$ and BA100_5 denotes a BA graph with 100 nodes and 5 edges addition per node. Also, we used 10 real-world graphs from Network Repository [31], including citation networks, web graphs, bio graphs, and road map graphs, all of which were handled as unlabeled and undirected graphs. To see all the results for these 10 graphs, please refer to Appendix D. For MINIMUMVERTEXCOVER and MAXIMUMINDEPENDENTSET, we additionally tested on DIMACS [3], difficult artificial instances. We generated other synthetic instances in Section 4.1.

## 4.1 Comparison of Generalization Ability

We compared the generalization ability of CombOpt Zero and S2V-DQN to various kinds of graphs. To see the pure contribution of CombOpt Zero, here CombOpt Zero incorporated the same graph representation model as S2V-DQN, namely S2V. Table 1 shows the performance for MINIMUMVERTEXCOVER and MAXCUT on various graph instances (see Appendix C for the explanation of each graph). While S2V-DQN had a better performance on ER graphs, which were used for training, CombOpt Zero showed a better generalization ability to the other synthetic graphs such as BA graphs, Watts-Strogatz graphs [37], and sparse regular graphs (i.e., graphs with the same degree of nodes). It was interesting that CombOpt Zero successfully learned the optimal solution of MAXCUT on trees (two-coloring of a tree puts all the edges into the cut set), while S2V-DQN does not. Appendix F visualizes how CombOpt Zero achieves the

[3]https://turing.cs.hbg.psu.edu/txn131/vertex_cover.html

Table 2: **GNN comparison for MINIMUMVERTEXCOVER.** Smaller is better. Since the inference takes $\Theta(n^3)$ per action for 2-IGN+, it did not finish within 2 hours for some test instances.

| | | | CombOpt Zero | | | | | |
|---|---|---|---|---|---|---|---|---|
| | $\|V\|$ | $\|E\|$ | 2-IGN+ | GIN | GCN | S2V | S2V-DQN | 2-approx |
| er100_15 | 100 | 783 | 77 | 77 | **76** | **76** | **76** | *83* |
| ba100_5 | 100 | 475 | 63 | 63 | 63 | 63 | 63 | *69* |
| cora | 2708 | 5429 | - | **1257** | 1258 | 1258 | 1258 | *1274* |
| citeseer | 3327 | 4552 | - | **1460** | 1462 | 1462 | 1461 | *1475* |
| web-edu | 3031 | 6474 | - | 1451 | 1451 | 1451 | 1451 | *1451* |
| web-spam | 4767 | 37375 | - | 2305 | **2298** | 2299 | 2319 | *2420* |
| soc-wiki-vote | 889 | 2914 | 413 | **406** | **406** | 407 | **406** | *406* |
| socfb-bowdoin47 | 2252 | 84387 | 2187 | 1793 | **1792** | 1796 | 1793 | *2052* |
| dimacs-frb30-15-1 | 450 | 17827 | 436 | 427 | **426** | **426** | 427 | *436* |
| dimacs-frb50-23-1 | 1150 | 80072 | 1132 | 1115 | **1111** | 1116 | 1114 | *1130* |

Table 3: **GNN comparison for MAXCUT.** Larger is better. CombOpt Zero outperformed the SOTA heuristic solvers on several real-world instances.

| | | | CombOpt Zero | | | | | | Heuristics | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $\|V\|$ | $\|E\|$ | 2-IGN+ | GIN | GCN | S2V | S2V-DQN | SDP | Burer | Laguna |
| er100_15 | 100 | 783 | 516 | 526 | 390 | 494 | **527** | *521* | *528* | *528* |
| ba100_5 | 100 | 475 | 324 | **343** | 282 | 337 | **343** | 341 | *344* | *344* |
| cora | 2708 | 5429 | - | **4268** | 3945 | 4260 | 4243 | - | *4394* | *4383* |
| citeseer | 3327 | 4552 | - | 3929 | 3477 | **3933** | 3893 | - | *3927* | *3927* |
| web-edu | 3031 | 6474 | - | 4705 | 4243 | **4712** | 4289 | - | *4679* | *4714* |
| web-spam | 4767 | 37375 | - | **23882** | 20498 | 20645 | 21027 | - | *25001* | *25070* |
| soc-wiki-vote | 889 | 2914 | 1645 | 2116 | 1730 | **2119** | 2064 | - | *2175* | *2163* |
| socfb-bowdoin47 | 2252 | 84387 | 41741 | **47426** | 20002 | 42063 | 37140 | - | *48639* | *48636* |

Table 4: **Improvement by test-time MCTS for MAXIMUMCLIQUE.** Larger is better. Results with test-time MCTS are shown in the parentheses. We broke the lower bounds of maximum clique size for some instances. Stability was also improved.

| | | | CombOpt Zero | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\|V\|$ | $\|E\|$ | 2-IGN+ | | GIN | | GCN | | S2V | | S2V-DQN | Best known |
| cora | 2708 | 5429 | 4 | (**5**) | 4 | (**5**) | 3 | (**5**) | 4 | (**5**) | 4 | *5* |
| citeseer | 3327 | 4552 | 4 | (6) | 5 | (6) | 4 | (6) | 4 | (6) | 4 | *9* |
| web-edu | 3031 | 6474 | 16 | (**30**) | 16 | (**30**) | 16 | (16) | 16 | (**30**) | 16 | *16* |
| web-spam | 4767 | 37375 | 10 | (**20**) | 16 | (17) | 7 | (17) | 16 | (17) | 16 | *14* |
| soc-wiki-vote | 889 | 2914 | 5 | (**7**) | 6 | (**7**) | 6 | (**7**) | 6 | (**7**) | 6 | *6* |
| socfb-bowdoin47 | 2252 | 84387 | 14 | (**23**) | 15 | (**23**) | 7 | (22) | 13 | (**23**) | 14 | *9* |

optimal solution on trees. CombOpt Zero also generalized better to real-world graphs although S2V-DQN performed better on a few instances.

## 4.2 Combination with Several GNN Models

Tables 2 and 3 show the comparison of performance among CombOpt Zero with four different GNN models (2-IGN+, GIN, GCN, and S2V) and S2V-DQN. In both MINIMUMVERTEXCOVER and MAXCUT, the use of recent GNN models enhanced the performance, but the best models were different across the problems and test instances. While GIN had the best performance in MAXCUT, GCN performed slightly better in MINIMUMVERTEXCOVER. See Appendix D for all the results of the 5 problems. One interesting insight was that GCN performed significantly worse in MAXCUT, while it did almost the best for the other four NP-hard problems. Overall, CombOpt Zero outperformed S2V-DQN by properly selecting a GNN model.

## 4.3 Test-time MCTS

The greedy selection in Sections 4.1 and 4.2 does not make full use of CombOpt Zero. When more computational time is allowed in test-time, CombOpt Zero can explore better solutions using the MCTS. Since the MCTS on large graphs takes a long time, we chose MAXIMUMCLIQUE as a case study because solution sizes and the MCTS depths are much smaller than the other problems. We used the same algorithm as in the training (Algorithm 1) with the same iteration coefficient ($c_{\text{iter}} = 4$) and $\tau = 0$, and selected an action based on the enhanced policy $\boldsymbol{\pi}$. In all the instances, the MCTS finished in a few minutes on a single process and single GPU, thanks to the small depth of the MCTS tree. The improvements are shown in Table 4. The test-time MCTS strictly improved performance on 7 instances out of 10 and got at least the same on all of the instances. Surprisingly, our results surpassed the best known solution reported in Network Repository on 6 out of 10 instances. For example, we found a clique of size 30 on web-edu, whose previous bound was 16.

### 4.4 Tradeoffs between CombOpt Zero and S2V-DQN

Here, we summarize some characteristics of CombOpt Zero and S2V-DQN. During the training, CombOpt Zero used 32 processes and four GPUs as described in Appendix C, while S2V-DQN used a single process and GPU because of its implementation. CombOpt Zero takes a longer time to generate self-play data than S2V-DQN due to the MCTS process. For this reason, CombOpt Zero needs a more powerful environment to obtain stable training. On the other hand, since CombOpt Zero is much more sample-efficient than S2V-DQN (see Appendix D.6), the bottleneck of the CombOpt Zero training is the data generation by the MCTS. This means that it can be highly optimized with an enormous GPU or TPU environment as in Silver et al. [34], Silver et al. [33], and Tian et al. [35].

By its nature, S2V-DQN can be also combined with other GNN models than S2V. However, S2V-DQN is directly implemented with GPU programming, it is practically laborious to combine various GNN models. On the other hand, CombOpt Zero is based on PyTorch framework [30] and it is relatively easy to implement different GNN models.

## 5 Conclusion

In this paper, we presented a general framework, CombOpt Zero, to solve combinatorial optimization on graphs without domain knowledge. The Monte Carlo Tree Search (MCTS) in training time successfully helped the wider exploration than the existing method and enhanced the generalization ability to various graphs. Combined with the recently-designed powerful GNN models, CombOpt Zero achieved even better performance. We also observed that the test-time MCTS significantly enhanced its performance and stability.

## Acknowledgements

MS was supported by the International Research Center for Neurointelligence (WPI-IRCN) at The University of Tokyo Institutes for Advanced Study.

## References

[1] Takuya Akiba and Yoichi Iwata. Branch-and-reduce exponential/fpt algorithms in practice: A case study of vertex cover. *Theoretical Computer Science*, 609:211–225, 2016.

[2] Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics*, 12(3):289–297, 1999.

[3] Barabási and Albert. Emergence of scaling in random networks. *Science*, 286 5439:509–12, 1999.

[4] Richard Bellman and Ronald Albert Howard. *Dynamic Programming and Markov Processes.* John Wiley, 1960.

[5] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *ArXiv*, abs/1611.09940, 2016.

[6] Samuel Burer, Renato Monteiro, and Yin Zhang. Rank-two relaxation heuristics for max-cut and other binary quadratic programs. *SIAM Journal on Optimization*, 12, 07 2001. doi: 10.1137/S1052623400382467.

[7] Stephen A. Cook. The complexity of theorem-proving procedures. In *ACM Symposium on Theory of Computing*, 1971.

[8] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Parameterized algorithms. In *Springer International Publishing*, 2015.

[9] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *International Conference on Machine Learning*, 2016.

[10] Iain Dunning, Swati Gupta, and John Silberholz. What works best when? a systematic evaluation of heuristics for max-cut and qubo. *INFORMS Journal on Computing*, 30:608–624, 08 2018. doi: 10.1287/ijoc.2017.0798.

[11] Paul Erdos. On random graphs. *Publicationes mathematicae*, 6:290–297, 1959.

[12] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, 2017.

[13] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42:1115–1145, 1995.

[14] Teofilo F. Gonzalez. *Handbook of Approximation Algorithms and Metaheuristics (Chapman & Hall/Crc Computer & Information Science Series)*. Chapman & Hall/CRC, 2007. ISBN 1584885505.

[15] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2019. URL http://www.gurobi.com.

[16] John J. Hopfield and David W. Tank. “neural” computation of decisions in optimization problems. *Biological Cybernetics*, 52:141–152, 1985.

[17] Nicolas Keriven and Gabriel Peyré. Universal invariant and equivariant graph neural networks. In *Neural Information Processing Systems*, 2019.

[18] Elias Boutros Khalil, Hanjun Dai, Yuyu Zhang, Bistra N. Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Neural Information Processing Systems*, 2017.

[19] Subhash Khot and Nisheeth K Vishnoi. On the unique games conjecture. In *Foundations of Computer Science*, volume 5, page 3, 2005.

[20] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[21] Levente Kocsis and Cs. Szepesvari. Bandit based monte-carlo planning. In *European Conference on Machine Learning*, 2006.

[22] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2018.

[23] Manuel Laguna, Abraham Duarte, and Rafael Marti. Hybridizing the cross-entropy method: An application to the max-cut problem. *Computers & Operations Research*, 36:487–498, 02 2009. doi: 10.1016/j.cor.2007.10.001.

[24] 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.

[25] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *Neural Information Processing Systems*, 2018.

[26] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. *ArXiv*, abs/1812.09902, 2018.

[27] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In *Neural Information Processing Systems*, 2019.

[28] Haggai Maron, Ethan Fetaya, Nimrod Segol, and Yaron Lipman. On the universality of invariant networks. In *International Conference on Machine Learning*, 2019.

[29] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Association for the Advancement of Artificial Intelligence*, 2018.

[30] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Neural Information Processing Systems*, pages 8024–8035. Curran Associates, Inc., 2019.

[31] Ryan A. Rossi and Nesreen Ahmed. The network data repository with interactive graph analytics and visualization. In *Association for the Advancement of Artificial Intelligence*, 2015.

[32] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[33] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *ArXiv*, abs/1712.01815, 2017.

[34] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[35] Yuandong Tian, Jerry Ma, Qucheng Gong, Shubho Sengupta, Zhuoyuan Chen, James Pinkerton, and C. Lawrence Zitnick. Elf opengo: An analysis and open reimplementation of alphazero. In *International Conference on Machine Learning*, 2019.

[36] Po-Wei Wang, Priya L. Donti, Bryan Wilder, and J. Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *International Conference on Machine Learning*, 2019.

[37] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of ‘small-world’ networks. *Nature*, 393:440–442, 1998.

[38] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *ArXiv*, abs/1810.00826, 2018.

## A Other NP-hard Problems

Here, we introduce two more NP-hard problems we used in the experiments.

### A.1 Maximum Independent Set

A subset of nodes $V' \subset V$ is called an independent set if no two nodes in $V'$ are adjacent; for all $(x, y) \in E$, $x \notin V'$ or $y \notin V'$ holds. MAXIMUMINDEPENDENTSET asks for an independent set whose size is maximum.

**MDP Formulation** $d$, $A_s$, Init, and $S_{\text{end}}$ are the same as MINIMUMVERTEXCOVER. $T(s, x)$ returns the next state, corresponding to the graph where $x$ and its adjacent nodes are deleted. $R(s, x) = 1$ because we want to maximize the number of transition steps of the MDP.

### A.2 Minimum Feedback Vertex Set

A subset of nodes $V' \subset V$ is called a feedback vertex set if the induced graph $G[V \backslash V']$ is cycle-free. MINIMUMFEEDBACKVERTEXSET asks a feedback vertex set whose size is minimum.

**MDP Formulation** $d$, $A_s$, $T(s, x)$, $R(s, x)$ and Init are the same as MINIMUMVERTEXCOVER. $S_{\text{end}}$ is the states with cycle-free graphs.

## B Graph Neural Networks

In this section, we review several Graph Neural Network (GNN) models and explain some modifications for our problem setting. Given a graph and input node feature $H^{(0)} \in \mathbb{R}^{n \times C_0}$, we aim at obtaining $y \in \mathbb{R}^{n \times C_{\text{out}}}$ which represents node feature as an output. $L$ denotes a number of layers.

### B.1 structure2vec

In `structure2vec` (S2V) [9], the feature is propagated by

$$H_v^{(l+1)} = \text{relu}\Big(\theta_1 H_v^{(0)} + \theta_2 \sum_{u \in \mathcal{N}(v)} H_u^{(l)}\Big),$$

where $\theta_1 \in \mathbb{R}^{p \times C_0}$, $\theta_2 \in \mathbb{R}^{p \times p}$ for some fixed integer $p$. The edge propagation term is ignored since we don't handle weighted edges in this work. In the last layer, the features of every node are aggregated to encode the whole graph:

$$y_v = \theta_3^T \text{relu}\big(\big[\theta_4 \sum_{u \in V} H_u^{(L)}, \theta_5 H_v^{(L)}\big]\big),$$

where $\theta_3 \in \mathbb{R}^{C_{\text{out}} \times 2p}$, $\theta_4, \theta_5 \in \mathbb{R}^{p \times p}$ and $[\cdot, \cdot]$ is the concatenation operator.

### B.2 Graph Convolutional Network

Graph Convolutional Network (GCN) [20] follows the layer-wise propagation rule:

$$H^{(l+1)} = \sigma(\tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} H^{(l)} \theta^{(l)}). \tag{8}$$

$\tilde{A} = A + I_n$ where $I_n$ is the identity matrix of the size $n$ and expresses an adjacency matrix with self-connections. Multiplying $H^{(l)}$ by $\tilde{A}$ means passing each node's feature vector to its neighbors' feature vectors in the next layer. $\tilde{D}$ is the degree matrix of $\tilde{A}$, such that $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ for diagonal elements and 0 for the other elements. $\tilde{D}^{-1/2}$ is multiplied to normalize $\tilde{A}$. $\theta^l$ is a trainable weight matrix in $l$-th layer.

### B.3 Graph Isomorphism Network

The propagation rule of Graph Isomorphism Network (GIN) [38] is as follows:

$$H^{(l+1)} = \text{MLP}^{(l)}(\tilde{A} H^{(l)}),$$

where $\text{MLP}^l$ refers to the multi-layer perceptron in the $l$-th layer. It takes the simple sum of features among the neighbors and itself by multiplying $\tilde{A}$. We adopt a similar suffix as the original paper:

$$y_v = \text{MLP}(\text{CONCAT}(H_v^{(l)} \mid l = 0, 1, \cdots, L)) \tag{9}$$

### B.4 2-IGN+

Different from message passing GNNs, each block of 2-IGN+ [27] takes a tensor as an input. It follows the propagation rule:

$$\mathbf{X}^{(l+1)} = B_l(\mathbf{X}^{(l)}). \tag{10}$$

$\mathbf{X}^{(l)} \in \mathbb{R}^{n\times n\times C_l}$ is a hidden tensor of the $l$-th block. $\mathbf{X}^{(0)} \in \mathbb{R}^{n\times n\times (C_0+1)}$ is initialized as the concatenation of the adjacency matrix and a tensor with node feature in its diagonal elements (other elements are 0). $B_l$ is the $l$-th block of 2-IGN+ and consists of three MLPs: $m_1^{(l)}, m_2^{(l)} \in \mathbb{R}^{C_l} \to \mathbb{R}^{C'_l}$ and $m_3^{(l)} \in \mathbb{R}^{C_l+C'_l} \to \mathbb{R}^{C'_l}$. After applying the two MLPs independently, we perform feature-wise matrix multiplication: $\mathbf{W}^{(l)}_{:,:,j} = m_1^{(l)}(\mathbf{X}^{(l)})_{:,:,j} \cdot m_2^{(l)}(\mathbf{X}^{(l)})_{:,:,j}$. The output of the block is the last MLP over the concatenation with $\boldsymbol{X}^{(l)}$: $\boldsymbol{X}^{(l+1)} = m_3^{(l)}([\mathbf{X}^{(l)}, \mathbf{W}^{(l)}])$. In our problem setting, we adopt equivariant linears layer instead of invariant linear layers to obtain the output tensor $y \in \mathbb{R}^{n\times C_{\text{out}}}$. We also adopt the suffix mentioned in the original paper:

$$y = \sum_{l=1}^{L} h^{(l)}(X^{(l)}), \tag{11}$$

where $h^{(l)} \in \mathbb{R}^{n\times C_l} \to \mathbb{R}^{n\times C_{\text{out}}}$ is an equivariant linear layer of $l$-th block.

## C Experiment Settings

In this section, we explain the detailed settings of experiments.

### C.1 Environment

The experiments were run on Intel Xeon E5-2695 v4, with four NVIDIA Tesla P100 GPUs. Libraries were compiled with GCC 5.4.0 and CUDA 9.2.148.

### C.2 Competitors

Our main competitor was S2V-DQN, the state-of-the-art reinforcement learning solver of NP-hard problems. We also tested some heuristics and approximation algorithms for specific problems. Note that a fair setting of the running environment is difficult since they do not use GPUs nor training time. Therefore, they were referred to just for checking how successfully CombOpt Zero learned combinatorial structure and algorithms for each problem. For MINIMUMVERTEXCOVER, MAXIMUMINDEPENDENTSET, MINIMUMFEEDBACKVERTEXSET and MAXCUT, we compared our algorithm to randomized algorithms. The results of the randomized algorithm were the best objective among 100 runs. For MAXCUT, we tried two heuristics solvers from MQLib [10]. We set the time limit of 10 minutes for these algorithms and used the best found solution as the results. We also used CPLEX to solve the integer programming formulation of MINIMUMVERTEXCOVER and MAXIMUMINDEPENDENTSET. Only CPLEX was run on MacBook Pro 2.4 GHz Quad-Core Intel Core i5. It was executed on 8 threads with the time limit of 10 minutes.

### C.3 Hyperparameters

Some of the hyperparameters are summarized in Table 5.

**Graph Neural Networks** For S2V, we used the same hyperparameters as the ones used in S2V-DQN [18]: we set embedding dimension $p = 64$ and the number of iterations $L = 5$. For GCN, we used a 5-layer network with a hidden dimension of size 32. For GIN, the network consisted of 5 layers of hidden dimension of size 32. Each MLP consisted of 5 layers also and the size of a hidden dimension was 16. Lastly, for 2-IGN+, we used a 2-block network with a hidden dimension of size 8. Each MLP had 2 layers with a hidden dimension of size 8.

**MCTS** We followed the implementation of AlphaGo Zero [34] and ELF [35]. We set $c_{\text{puct}} = 1.5$. In training phase and test-time MCTS, we added Dirichlet noise $\boldsymbol{\eta} \sim \text{Dir}(\mathbf{0.03})$ to the first move to explore a variety of actions. More specifically, the policy $\boldsymbol{p}$ for the first action is modified to $\boldsymbol{p} \leftarrow (1-\varepsilon)\boldsymbol{p} + \varepsilon\boldsymbol{\eta}$, where $\varepsilon = 0.25$. We set $c_{\text{iter}} = 4$, i.e., ran MCTS for $c_{\text{iter}}|A_{s_0}|$ times, for MAXCUT, MAXIMUMCLIQUE, MAXIMUMINDEPENDENTSET. We set $c_{\text{iter}} = 3$ for MINIMUMVERTEXCOVER and MINIMUMFEEDBACKVERTEXSET because they took relatively longer time due to larger solution sizes. We set the temperature as $\tau = 1$ during the training phase, and $\tau = 0$ in the test-time MCTS. $\tau = 0$ means to take an action whose visit count is the maximum (if there are multiple possible actions, choose one at

Table 5: **Part of hyperparameters.** For each of the five problems, some important hyperparameters are summarized. $\min n$, $\max n$, $p$ regard ER graphs, $c_{\text{iter}}$ is for MCTS, and $\text{keep}$ denotes the duration (minutes) to keep trajectories.

| | $\min n$ | $\max n$ | $p$ | $c_{\text{iter}}$ | keep |
|---|---|---|---|---|---|
| MVC | 80 | 100 | 0.15 | 3 | 10 |
| MAXCUT | 40 | 50 | 0.15 | 4 | 5 |
| MAXCLIQUE | 80 | 100 | 0.5 | 4 | 5 |
| MIS | 80 | 100 | 0.15 | 4 | 5 |
| FVS | 80 | 100 | 0.15 | 3 | 10 |

random). When a new node is visited in MCTS, we calculated the mean and standard deviation of the reward from its corresponding state. We approximated these value from 20 random plays.

**Training** Each *learner* sampled 20 trajectories from the self-play records and ran stochastic gradient descent by Adam, where the learning rate is 0.001, weight decay is 0.0001, and batch size is 16. After repeating this routine for 15 times, the *learner* saved the new model. The number of *data generators* was 24 except when 2-IGN+ was used. In that case, the number was 20. The number of *learners* and *model evaluators* were set to 6 and 2, respectively. Each *model evaluator* generated 50 test ER graphs with $n = 100$ and $p = 0.15$ ($n = 50$ for MAXCUT and $p = 0.5$ for MAXIMUMCLIQUE) each time. It compared the performance of the best model and a newly generated model by the cummulative objective and managed the best model. Each trajectory was removed after 5 minutes (10 minutes for MINIMUMVERTEXCOVER and MINIMUMFEEDBACKVERTEXSET) since it was generated.

## D Performance Comparison of CombOpt Zero and Other Approaches

In this section, we show the full results of our experiments. We compared CombOpt Zero to some simple randomized algorithms and state-of-the-art solvers, in addition to S2V-DQN. For each of the five problems, we first explain the characteristics of the problem and some famous approaches, then we show the comparison of the performances.

### D.1 Minimum Vertex Cover

**Approximability** There is a simple 2-approximation algorithm for MINIMUMVERTEXCOVER. It greedily obtains a maximal (not necessarily maximum) matching and outputs the nodes in the matching as a solution. Under Unique Games Conjecture [19], MINIMUMVERTEXCOVER cannot be approximated better than this.

**Randomized Heuristics** Regardless of the hardness of approximation of MINIMUMVERTEXCOVER, there are some practical algorithms to solve this (and equivalently, MAXIMUMINDEPENDENTSET) [1, 24]. These state-of-the-art algorithms usually iteratively *kernelize* the graph, i.e., reduce the size of the problem, and search better solutions, either in an exact way or in an approximated way. We adopted the easiest reduction rule to design a simple randomized algorithm (Algorithm 3) for MINIMUMVERTEXCOVER; if the input graph has a degree-1 vertex $v$, there is always a vertex cover that does not contain $v$. Although theoretically, this reduction rule does not improve the approximation ratio, i.e., the approximation is still 2, it is effective because we can cut off trivial solutions and make the size of the problem smaller.

**Full results** Table 6 shows the full results for MINIMUMVERTEXCOVER. Although both CombOpt Zero and S2V-DQN failed to find an optimal solution for the ER graph of 200 nodes (er200_10), both of them found near-optimal solutions for large cases even though they were trained on small random graphs. Remarkably, some models of CombOpt Zero found better solutions than CPLEX for large random graphs. However, for some large sparse real-world networks, they performed worse than a simple 2-approximation algorithm. A hybrid approach that mixes the reduction rules and machine learning, as done in [25], may be effective in such sparse networks, but our work focuses on a method without domain knowledge. Theoretically, 2-IGN+ is stronger than GIN or GCN in terms of discriminative power [27], its performance on large instances was not good although they successfully learned er100_15. One reason could be because we reduced the size of the network to fasten the training. Also, since 2-IGN+ is not a message passing neural network, its tendency of learning could be different from other GNNs. We leave the empirical and theoretical analyses on the characteristics of learning by 2-IGN+ as future work.

**Algorithm 3** A Simple Randomized Algorithm for MINIMUMVERTEXCOVER

```
Require: Graph $G = (V, E)$
Ensure: Size of a vertex cover $r$
  $r \leftarrow 0$
  while $G$ is not empty do
    if $G$ has a degree-1 vertex then
      $v \leftarrow$ uniformly randomly chosen degree-1 vertex
    else
      $v \leftarrow$ uniformly randomly chosen vertex
    end if
    $r \leftarrow r + 1$
    $G \leftarrow$ delete $v$ and its neighbor(s)
  end while
```

Table 6: **Performance comparison on** MINIMUMVERTEXCOVER. Smaller is better. Bold values are the best values among reinforcement learning approaches (CombOpt Zero and S2V-DQN). Since the inference takes $\Theta(n^3)$ per action for 2-IGN+, it did not finish testing within 2 hours for some instances.

| | | | CombOpt Zero | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | $\|V\|$ | $\|E\|$ | 2-IGN+ | GIN | GCN | S2V | S2V-DQN | 2-approx | CPLEX |
| er100_15 | 100 | 783 | 77 | 77 | **76** | **76** | **76** | *83* | *76* |
| er200_10 | 200 | 1957 | **160** | **160** | 161 | **160** | **160** | *174* | *159* |
| er1000_5 | 1000 | 25091 | 907 | 900 | **893** | 900 | 898 | *954* | *894* |
| er5000_1 | 5000 | 124804 | - | 4479 | **4448** | 4484 | 4482 | *4793* | *4480* |
| ba100_5 | 100 | 475 | 63 | 63 | 63 | 63 | 63 | *69* | *63* |
| ba200_5 | 200 | 975 | 121 | 120 | 120 | 120 | 120 | *134* | *118* |
| ba1000_5 | 1000 | 4975 | 745 | 592 | **591** | 592 | 594 | *670* | *589* |
| ba5000_5 | 5000 | 24975 | - | 2920 | **2905** | 2920 | 2927 | *3374* | *2932* |
| cora | 2708 | 5429 | - | **1257** | 1258 | 1258 | 1258 | *1274* | *1257* |
| citeseer | 3327 | 4552 | - | **1460** | 1462 | 1462 | 1461 | *1475* | *1460* |
| web-edu | 3031 | 6474 | - | 1451 | 1451 | 1451 | 1451 | *1451* | *1451* |
| web-spam | 4767 | 37375 | - | 2305 | **2298** | 2299 | 2319 | *2420* | *2297* |
| road-minnesota | 2642 | 3303 | - | **1322** | 1323 | 1324 | 1329 | *1330* | *1319* |
| bio-yeast | 1458 | 1948 | 702 | **456** | **456** | **456** | 457 | *456* | *456* |
| bio-SC-LC | 2004 | 20452 | 1471 | 1041 | 1046 | **1039** | 1053 | *1245* | *1036* |
| rt_damascus | 3052 | 3881 | 370 | 369 | 369 | 369 | 369 | *369* | *369* |
| soc-wiki-vote | 889 | 2914 | 413 | **406** | **406** | 407 | **406** | *406* | *406* |
| socfb-bowdoin47 | 2252 | 84387 | 2187 | 1793 | **1792** | 1796 | 1793 | *2052* | *1792* |
| dimacs-frb30-15-1 | 450 | 17827 | 436 | 427 | **426** | **426** | 427 | *436* | *421* |
| dimacs-frb50-23-1 | 1150 | 80072 | 1132 | 1115 | **1111** | 1116 | 1114 | *1130* | *1107* |

## D.2 Max Cut

**Approximability** MAXCUT has a famous 0.878-approximation randomized algorithm by SDP (semidefinite programming) [13]. Similarly to MINIMUMVERTEXCOVER, this approximation ratio is best under Unique Games Conjecture [19].

**Competing Algorithms** Although the randomized algorithm mentioned above has a polynomial-time complexity and the best approximation ratio, they are rarely applied for graphs of thousands of nodes due to the large size of the SDP. We only applied this algorithm to graphs that have no more than 200 nodes. Instead, we compared the performance with a MAXCUT solver by Dunning et al. [10].

**Full results** Table 7 shows the full results for MAXCUT. We compared CombOpt Zero, S2V-DQN, the SDP approximation algorithm and heuristics. Unlike MINIMUMVERTEXCOVER, CombOpt Zero with GCN had poor performance. CombOpt Zero with GIN had the best performance and it overperformed S2V-DQN. CombOpt Zero even overperformed state-of-the-art heuristics for some instances such as citeseer and rt_damascus. It is also remarkable that all of the results by CombOpt Zero with GIN overperformed 0.878-approximation SDP algorithm (note that SDP did not finish for large instances).

Table 7: **Performance comparison on** MAXCUT. Larger is better. Bold values are the best values among reinforcement learning approaches. Empty cells mean time limit exceeded.

| | | | CombOpt Zero | | | | | | Heuristics | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $\|V\|$ | $\|E\|$ | 2-IGN+ | GIN | GCN | S2V | S2V-DQN | SDP | Burer | Laguna |
| er100_15 | 100 | 783 | 516 | 526 | 390 | 494 | **527** | *521* | *528* | *528* |
| er200_10 | 200 | 1957 | 1194 | **1269** | 974 | 1221 | 1262 | *1266* | *1289* | *1289* |
| er1000_5 | 1000 | 25091 | 12278 | **14787** | 8596 | 12561 | 14424 | - | *15164* | *15140* |
| er5000_1 | 5000 | 124804 | - | **73275** | 42860 | 63389 | 71909 | - | *75601* | *75406* |
| ba100_5 | 100 | 475 | 324 | **343** | 282 | 337 | **343** | 341 | *344* | *344* |
| ba200_5 | 200 | 975 | 639 | 694 | 576 | 687 | **698** | *693* | *703* | *703* |
| ba1000_5 | 1000 | 4975 | 2595 | 3485 | 2941 | **3492** | 3465 | - | *3589* | *3580* |
| ba5000_5 | 5000 | 24975 | - | **17643** | 15015 | 17381 | 16870 | - | *17997* | *17911* |
| cora | 2708 | 5429 | - | **4268** | 3945 | 4260 | 4243 | - | *4394* | *4383* |
| citeseer | 3327 | 4552 | - | 3929 | 3477 | **3933** | 3893 | - | *3927* | *3927* |
| web-edu | 3031 | 6474 | - | 4705 | 4243 | **4712** | 4289 | - | *4679* | *4714* |
| web-spam | 4767 | 37375 | - | **23882** | 20498 | 20645 | 21027 | - | *25001* | *25070* |
| road-minnesota | 2642 | 3303 | - | 3079 | 2856 | **3080** | 3015 | - | *3091* | *3024* |
| bio-yeast | 1458 | 1948 | 714 | **1769** | 1582 | **1769** | 1751 | - | *1770* | *1761* |
| bio-SC-LC | 2004 | 20452 | 7967 | **14358** | 11589 | 10893 | 11890 | - | *14586* | *14583* |
| rt_damascus | 3052 | 3881 | - | 3694 | 3439 | **3698** | 3667 | - | *3617* | *3683* |
| soc-wiki-vote | 889 | 2914 | 1645 | 2116 | 1730 | **2119** | 2064 | - | *2175* | *2163* |
| socfb-bowdoin47 | 2252 | 84387 | 41741 | **47426** | 20002 | 42063 | 37140 | - | *48639* | *48636* |

### D.3 Maximum Clique

For MAXIMUMCLIQUE, we compared CombOpt Zero and S2V-DQN. For CombOpt Zero, since the solution sizes were relatively small, we also tested test-time MCTS.

Although MAXIMUMCLIQUE is equivalent to MAXIMUMINDEPENDENTSET on complement graphs, since the number of edges in the complement graphs of sparse networks becomes too large, it is usually difficult to solve MAXIMUM-CLIQUE by algorithms designed for MINIMUMVERTEXCOVER.

**Full results** Full results for MAXIMUMCLIQUE is given in Table 8. CombOpt Zero with GIN worked the best among four models and outperformed S2V-DQN especially on random graphs. We can also observe that the performance is strongly enhanced by test-time MCTS. For example, although the performance of CombOpt Zero with GCN or 2-IGN+ was relatively poor than other GNNs, by test-time MCTS, its performance became much stabler and solutions were also improved. Since test-time MCTS requires $n$ times larger time complexity, where $n$ is the number of nodes, it is hard to apply test-time MCTS on large instances when solution size is too large or time limit is too short.

### D.4 Maximum Independent Set

**Randomized Algorithm** We also compared our methods and S2V-DQN to a randomized algorithm for MAXIMU-MINDEPENDENTSET. We adopt the randomized algorithm developed in 3 with a slight change: for a degree-1 vertex $v$, there is always a maximum independent set that includes $v$. Therefore, instead, we include $v$ to the current solution and erase $v$ and its neighbors from the original graph. Note that although MINIMUMVERTEXCOVER and MAXIMUMINDE-PENDENTSET are theoretically equivalent, when solved by CombOpt Zero or S2V-DQN, their difficulties are different. For example, in the first stage of the training, the sizes of independent sets obtained are usually much greater than the complements of vertex covers (usually, they tend to select almost all nodes as vertex cover at the beginning).

**Full results** Please refer to Table 9 for the full results for MAXIMUMINDEPENDENTSET. Similarly to MINIMUMVER-TEXCOVER, even a simple randomized algorithm with recursive reductions could obtain near-optimal solutions for large sparse networks, its performance on ER graphs were much weaker than both CombOpt Zero and S2V-DQN. This is because, in ER graphs, there were only a few trivial vertices to be selected. Notably, CombOpt Zero with GIN reached to a better solution than CPLEX on one of the DIMACS instances.

### D.5 Minimum Feedback Vertex Set

**Randomized Algorithm** Similarly to MINIMUMVERTEXCOVER, a 2-approximation algorithm is known for MINI-MUMFEEDBACKVERTEXSET [2]. In this experiment, we implemented a parameterized algorithm that runs in time

Table 8: **Performance comparison on** MAXIMUMCLIQUE**.** Larger is better. Results with test-time MCTS are shown in the parentheses. Best known values were obtained from Network Repository [31]. Test-time MCTS broke sove lower bounds of maximum clique size for some instances.

| | | | CombOpt Zero | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\lvert V\rvert$ | $\lvert E\rvert$ | 2-IGN+ | | GIN | | GCN | | S2V | | S2V-DQN | Best known |
| er100_15 | 100 | 783 | **4** | (**4**) | **4** | (**4**) | **4** | (**4**) | **4** | (**4**) | 3 | - |
| er200_10 | 200 | 1957 | **4** | (**4**) | **4** | (**4**) | **4** | (**4**) | **4** | (**4**) | 3 | - |
| er1000_5 | 1000 | 25091 | **4** | (**4**) | **4** | (**4**) | **4** | (**4**) | **4** | (**4**) | 3 | - |
| er5000_1 | 5000 | 124804 | 3 | (**4**) | 3 | (**4**) | 3 | (**4**) | 3 | (**4**) | 3 | - |
| ba100_5 | 100 | 475 | **5** | (**5**) | **5** | (**5**) | **5** | (**5**) | **5** | (**5**) | 3 | - |
| ba200_5 | 200 | 975 | **5** | (**5**) | **5** | (**5**) | 4 | (**5**) | **5** | (**5**) | 3 | - |
| ba1000_5 | 1000 | 4975 | **5** | (**5**) | **5** | (**5**) | **5** | (**5**) | **5** | (**5**) | 3 | - |
| ba5000_5 | 5000 | 24975 | 4 | (**5**) | 4 | (**5**) | 4 | (**5**) | 4 | (**5**) | 3 | - |
| cora | 2708 | 5429 | 4 | (**5**) | 4 | (**5**) | 3 | (**5**) | 4 | (**5**) | 4 | **5** |
| citeseer | 3327 | 4552 | 4 | (6) | 5 | (6) | 4 | (6) | 4 | (6) | 4 | **9** |
| web-edu | 3031 | 6474 | 16 | (**30**) | 16 | (**30**) | 16 | (16) | 16 | (**30**) | 16 | 16 |
| web-spam | 4767 | 37375 | 10 | (**20**) | 16 | (17) | 7 | (17) | 16 | (17) | 16 | 14 |
| road-minnesota | 2642 | 3303 | 2 | (2) | 2 | (**3**) | 2 | (**3**) | **3** | (**3**) | **3** | **3** |
| bio-yeast | 1458 | 1948 | 3 | (**6**) | 4 | (**6**) | 2 | (**6**) | 3 | (**6**) | 4 | 5 |
| bio-SC-LC | 2004 | 20452 | 12 | (**29**) | **29** | (**29**) | 3 | (**29**) | **29** | (**29**) | **29** | **29** |
| rt_damascus | 3052 | 3881 | 3 | (**4**) | 3 | (**4**) | 3 | (**4**) | **4** | (**4**) | 3 | **4** |
| soc-wiki-vote | 889 | 2914 | 5 | (**7**) | 6 | (**7**) | 6 | (**7**) | 6 | (**7**) | 6 | 6 |
| socfb-bowdoin47 | 2252 | 84387 | 14 | (**23**) | 15 | (**23**) | 7 | (22) | 13 | (**23**) | 14 | 9 |

Table 9: **Performance comparison on** MAXIMUMINDEPENDENTSET**.** Larger is better. Bold values are the best values among reinforcement learning approaches. Empty cells mean time limit exceeded.

| | | | CombOpt Zero | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | $\lvert V\rvert$ | $\lvert E\rvert$ | 2-IGN+ | GIN | GCN | S2V | S2V-DQN | randomized | CPLEX |
| er100_15 | 100 | 783 | **24** | **24** | 23 | **24** | **24** | *23* | *24* |
| er200_10 | 200 | 1957 | 40 | 40 | 39 | **41** | 40 | *37* | *41* |
| er1000_5 | 1000 | 25091 | 106 | 107 | **108** | 105 | 106 | *89* | *107* |
| er5000_1 | 5000 | 124804 | - | 544 | 538 | 544 | **551** | *435* | *544* |
| ba100_5 | 100 | 475 | 37 | 37 | 37 | 37 | 37 | *37* | *37* |
| ba200_5 | 200 | 975 | 81 | 81 | 80 | **82** | **82** | *79* | *82* |
| ba1000_5 | 1000 | 4975 | 400 | 407 | 403 | 408 | **409** | *394* | *411* |
| ba5000_5 | 5000 | 24975 | - | 2079 | 2062 | **2085** | 2078 | *1960* | *2090* |
| cora | 2708 | 5429 | - | 1450 | 1448 | **1451** | 1448 | *1439* | *1451* |
| citeseer | 3327 | 4552 | - | 1818 | 1817 | **1819** | 1817 | *1860* | *1867* |
| web-edu | 3031 | 6474 | - | 1580 | 1580 | 1580 | 1580 | *1580* | *1580* |
| web-spam | 4767 | 37375 | - | **2464** | 2456 | 2463 | 2441 | *2434* | *2470* |
| road-minnesota | 2642 | 3303 | - | 1318 | 1300 | 1316 | **1321** | *1313* | *1323* |
| bio-yeast | 1458 | 1948 | 990 | 1000 | 1001 | **1002** | **1002** | *1002* | *1002* |
| bio-SC-LC | 2004 | 20452 | 945 | 959 | 953 | **964** | 948 | *936* | *968* |
| rt_damascus | 3052 | 3881 | - | 2673 | **2683** | 2679 | **2683** | *2683* | *2683* |
| soc-wiki-vote | 889 | 2914 | 481 | **483** | 482 | **483** | 482 | *483* | *483* |
| socfb-bowdoin47 | 2252 | 84387 | 445 | 456 | **457** | 443 | 426 | *392* | *461* |
| dimacs-frb30-15-1 | 450 | 17827 | 26 | 26 | 26 | **27** | 26 | *24* | *28* |
| dimacs-frb50-23-1 | 1150 | 80072 | 41 | **44** | 43 | 40 | 41 | *39* | *43* |

$4^k n^{O(1)}$ , where $k$ is the size of the solution, instead, which would work well for real-world sparse networks. The implemented algorithm is based on the following theorem.

**Theorem D.1** (Cygan et al. [8], page 101)**.** *Let $G = (V, E)$ be a multigraph whose minimum degree is at least 3. Then, for any feedback vertex set $V' \subset V$, more than half of edges have at least one endpoint in $V'$.*

**Corollary D.1.** *Let $G = (V, E)$ be a multigraph whose minimum degree is at least 3. If we choose a vertex $v$ with probability proportional to its degree, $v$ is included in a minimum feedback vertex set of G with probability greater than* $1/4$.

*Proof.* This follows from the fact that a uniformly randomly chosen edge in $G$ has at least one endpoint in a minimum feedback vertex set with probability more than $1/2$. ☐

**Algorithm 4** A Simple Randomized Algorithm for MINIMUMFEEDBACKVERTEXSET

```
Require: Graph $G = (V, E)$
Ensure: Size of a feedback vertex set $r$
  $r \leftarrow 0$
  while $G$ has cycle(s) do
    $(G, r) \leftarrow \text{reduce}(G, r)$
    $v \leftarrow$ select a random node by Corollary D.1
    $r \leftarrow r + 1$
    $G \leftarrow$ remove vertex $v$
  end while
```

Table 10: **Performance comparison on MINIMUMFEEDBACKVERTEXSET.** Smaller is better. Bold values are the best values among reinforcement learning approaches. Empty cells mean time limit exceeded.

| | | | CombOpt Zero | | | | | |
|---|---|---|---|---|---|---|---|---|
| | $\lvert V\rvert$ | $\lvert E\rvert$ | 2-IGN+ | GIN | GCN | S2V | S2V-DQN | randomized |
| er100_15 | 100 | 783 | 64 | **63** | 69 | **63** | **63** | *72* |
| er200_10 | 200 | 1957 | 143 | 137 | 150 | 137 | **136** | *156* |
| er1000_5 | 1000 | 25091 | 917 | **844** | 868 | 848 | 874 | *909* |
| er5000_1 | 5000 | 124804 | - | **4201** | 4339 | 4229 | 4359 | *4567* |
| ba100_5 | 100 | 475 | 44 | 40 | 44 | 38 | **37** | *46* |
| ba200_5 | 200 | 975 | 91 | 70 | 77 | **69** | **69** | *88* |
| ba1000_5 | 1000 | 4975 | 828 | 341 | 367 | **328** | 330 | *445* |
| ba5000_5 | 5000 | 24975 | - | 1753 | 1879 | 1692 | **1678** | *2265* |
| cora | 2708 | 5429 | - | 527 | 571 | 545 | **509** | *565* |
| citeseer | 3327 | 4552 | - | **447** | 471 | 457 | 450 | *430* |
| web-edu | 3031 | 6474 | - | 525 | **479** | 540 | 859 | *247* |
| web-spam | 4767 | 37375 | - | 1514 | 1587 | 1486 | **1461** | *1601* |
| road-minnesota | 2642 | 3303 | - | **401** | 523 | 478 | 415 | *316* |
| bio-yeast | 1458 | 1948 | 794 | 153 | 168 | 186 | **144** | *122* |
| bio-SC-LC | 2004 | 20452 | 1596 | 856 | 859 | **796** | 1591 | *895* |
| rt_damascus | 3052 | 3881 | 1424 | **120** | 150 | 132 | 286 | *98* |
| soc-wiki-vote | 889 | 2914 | 677 | 205 | 204 | 204 | **201** | *227* |
| socfb-bowdoin47 | 2252 | 84387 | 2066 | 1773 | 1741 | **1619** | 2175 | *1763* |

We iteratively reduce the input graph by following rules so that the minimum degree is at least 3.

1. If multigraph $G$ has a self loop at $v$, remove vertex $v$ and include $v$ to feedback vertex set.
2. If multigraph $G$ has multiedges whose multiplicity is greater than 2, reduce the multiplicity to 2.
3. If multigraph $G$ has degree 2 vertex $v$, erase $v$ and connect two neighbors of $v$ by a new edge.
4. If multigraph $G$ has degree-1 vertex $v$, remove vertex $v$.

Therefore, we can construct the following randomized parameterized algorithm (Algorithm 4) for MINIMUMFEEDBACKVERTEXSET. This algorithm provides a minimum feedback vertex set with a probability greater than $1/4^k$, where $k$ is the size of an optimal solution. Note that we do not have to recursively reduce the graph once the minimum degree becomes at least 3 to obtain this bound. However, practically, we can get much better solutions through recursive reductions.

**Full results** Full results for MINIMUMFEEDBACKVERTEXSET is given in Table 10. For the randomized parameterized algorithm, we ran it for 100 times and took the best objective as the score. Although in this problem, we could not observe a significant superiority of CombOpt Zero against S2V-DQN, we can see a significant difference of solution sizes for some of the datasets such as web-edu, bio-SC-LC, and socfb-bowdoin47. This is probably because all of the five S2V-DQN models converged into local optima. Interestingly even the performances were significantly poor on certain instances, they still sometimes overperformed CombOpt Zero on other instances. The randomized algorithm effectively worked on large real-world networks where the degrees were biased. On the other hand, on ER graphs, since the degrees of nodes were not as variant as social networks, CombOpt Zero and S2V-DQN performed much better than the randomized algorithm.

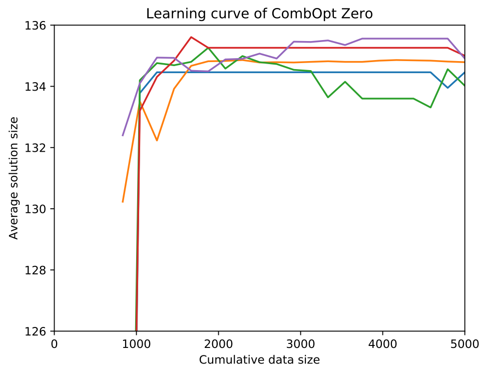

Figure 3: **Training curve for CombOpt Zero on MAXCUT.** Data for five models with the same hyperparameters are shown. The horizontal axis is the cumulative number of generated trajectories during 2 hours. Since, CombOpt Zero saves the best models after 15 minutes from start, its learning curve is not shown for the first 15 minutes.

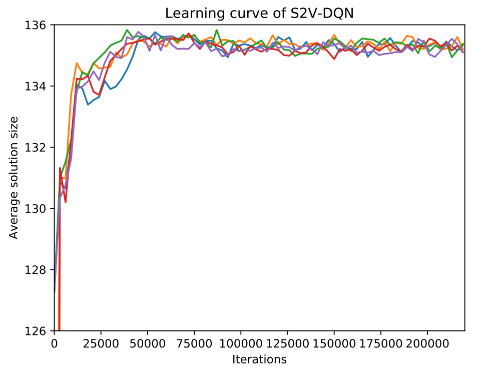

Figure 4: **Training curve for S2V-DQN on MAXCUT.** Data for five models with the same hyperparameters are shown. The horizontal axis shows the total number of generated trajectories during 2 hours (on average, S2V-DQN generates one trajectory per iteration).

### D.6 Training Convergence

Here, we show the learning curves of CombOpt Zero and S2V-DQN for MAXCUT. For CombOpt Zero, we chose S2V as the network. Figure 3 and Figure 4 are the learning curves for CombOpt Zero and S2V-DQN respectively. As explained in Section 4, we trained both CombOpt Zero and S2V-DQN for 2 hours. The horizontal axes show the number of trajectories generated during this 2 hours. The vertical axes correspond to the average cut size found on 100 fixed random ER graphs of size 50. Since CombOpt Zero starts to log the best models after 15 minutes from the beginning, and updates the log each 5 minutes after that, data for the first 15 minutes is not shown on the chart. Unlike S2V-DQN, if the newly generated models do not perform better than the current one, the best model is not updated and hence the learning curve remains flat for some intervals.

While S2V-DQN generated about 225000 data in two hours on a single GPU, CombOpt Zero generated only 5000 data on four GPUs. This is because the MCTS, which CombOpt Zero executes to generate self-play data, takes time. On the other hand, S2V-DQN requires about 50000 data for training to converge, while CombOpt Zero requires only about 2000 data, meaning that the training of CombOpt Zero is much more sample-efficient.

## E Comparison of Reduction Rules

Here, we discuss some differences of reinforcement learning formulation between S2V-DQN and CombOpt Zero. The first difference is their states. While S2V-DQN's state keeps both the original graph and selected nodes, CombOpt Zero's state is a single current (labeled) graph and its size usually gets smaller as the state proceeds. Since the GNN inference of smaller graphs is faster, it's more efficient to give small graphs as an input. Although S2V-DQN implicitly reduces the graph size in most of its implementation, our formulation is more explicit.

The second difference is that while S2V-DQN limits action space to a selection of one vertex, CombOpt Zero does not. Thanks to this, as stated in 3.1, MAXCUT can be formulated as a node coloring by two colors with more intuitive termination criteria: finish coloring all the nodes. This also allows the application to a wider range of problems. For example, K-COLORING by defining the action space by $\{(x, c) \mid x \in V, c \in \mathbb{N}, 0 \leq c \leq K\}$, where $(x, c)$ represents the coloring of node $x$ by color $c$.

## F Visualization

In this section, we illustrate how CombOpt Zero finds solutions for MAXCUT. As described in Section 4, CombOpt Zero found an optimal solution for trees and its overall performance was comparable to the state-of-the-art heuristic solvers. Figure 5 shows the sequence of actions by CombOpt Zero on an ER graph. Starting from one node, CombOpt Zero colors surrounding nodes one by one with the opposite color. Figure 6 shows the sequence of actions on a tree. The order of actions was similar to the order of visiting nodes in the depth-first-search. Since an optimal two coloring for MAXCUT on a tree can be obtained by the depth-first-search, we can say that CombOpt Zero successfully learned the effective algorithm for MAXCUT on a tree from the training on random graphs.

It is also interesting that CombOpt Zero sometimes skips the neighbors and colors a two-hop node by the same color as the current node. This is possible because the $L$-layer (5 in our case) message passing GNN can catch information of the $L$-hop neighbors. This flexibility possibly affected the good performance on other graph instances than trees.

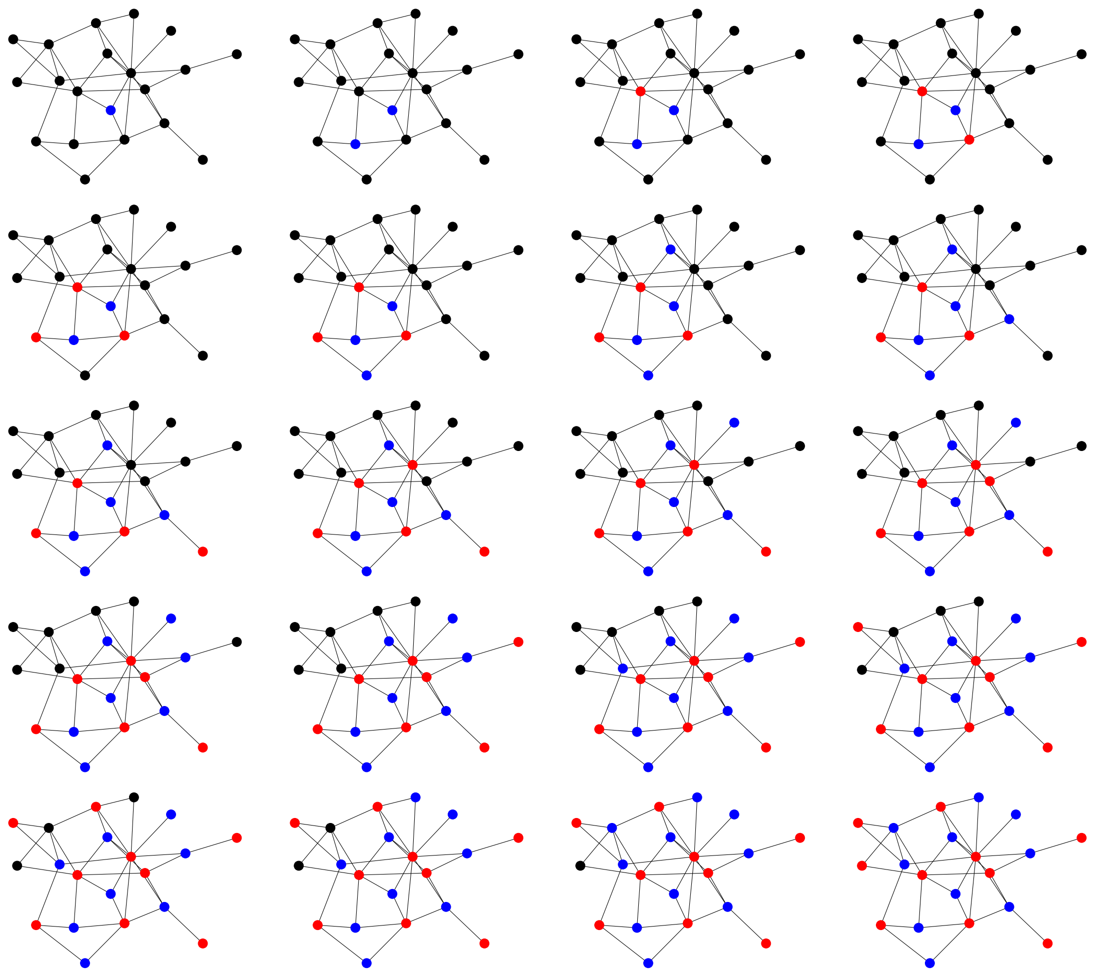

Figure 5: **Order of actions of CombOpt Zero for** MAXCUT **on a ER graph.** Although the order of selecting nodes and the coloring is arbitrary, CombOpt Zero learned to color neighbors one by one with the opposite color.

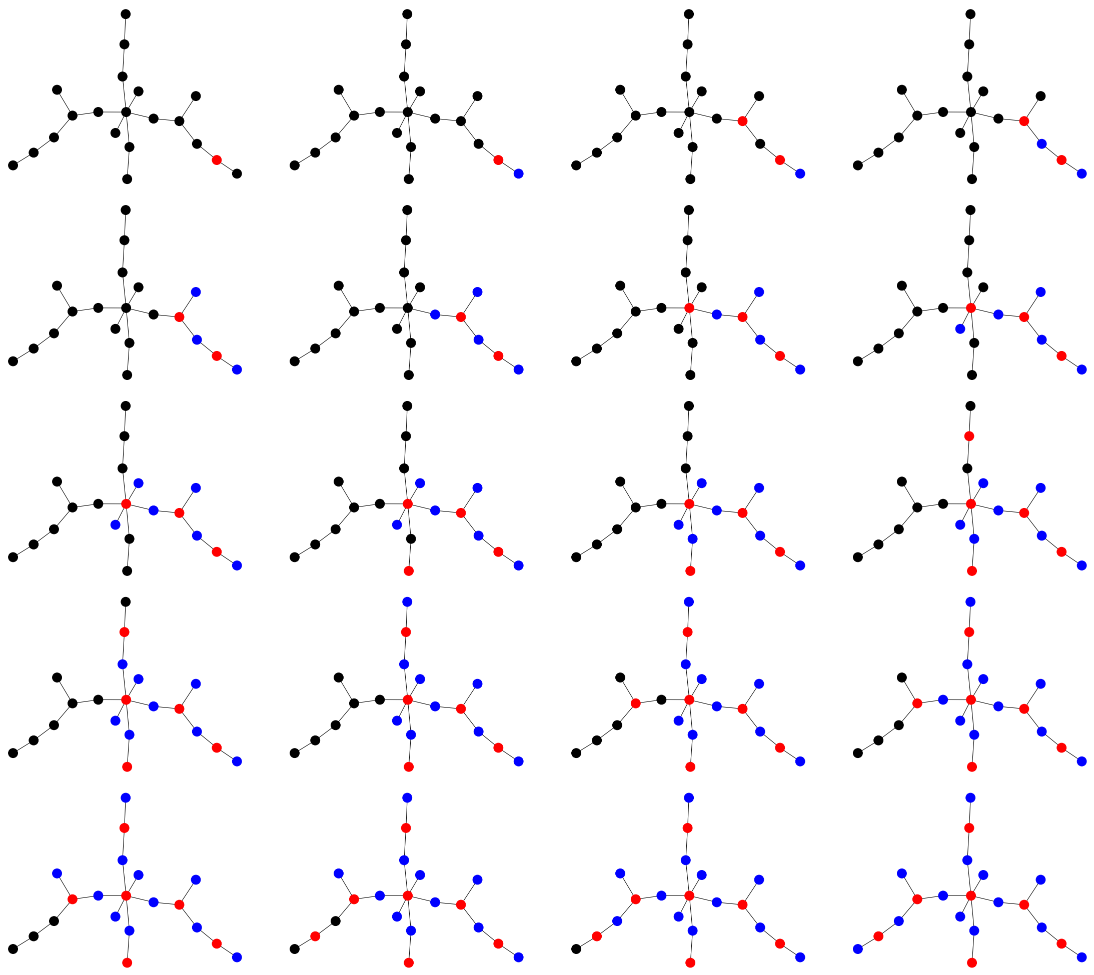

Figure 6: **Order of actions of CombOpt Zero for** MAXCUT **on a tree.** It successfully found an optimal solution. The order of actions is similar to the order of visiting nodes in the depth-first-search.