Maximum Independent Set: Self-Training through Dynamic Programming

Anonymous Author(s)
Affiliation
Address
email

Abstract

This work presents a novel graph neural network (GNN) framework for solving the maximum independent set (MIS) inspired by dynamic programming (DP). Specifically, given a graph, we propose a DP-like recursive algorithm based on GNNs that firstly constructs two smaller sub-graphs, predicts the one with the larger MIS, and then uses it in the next recursive call. To train our algorithm, we require annotated comparisons of different graphs concerning their MIS size. Annotating the comparisons with the output of our algorithm leads to a self-training process that results in more accurate self-annotation of the comparisons and vice versa. We provide numerical evidence showing the superiority of our method vs prior methods in multiple synthetic and real-world datasets.

1 Introduction

Deep neural networks (DNNs) have achieved unprecedented success in extracting intricate patterns directly from data without the need for handcrafted rules, while still generalizing well to new and previously unseen instances [He et al., 2016, Vaswani et al., 2017]. Among other applications, this success has led to the development of frameworks that utilize DNNs to solve combinatorial optimization (CO) problems, such as the Traveling Salesman Problem [Xing and Tu, 2020, Hu et al., 2020, Prates et al., 2019], the Job-Shop Scheduling Problem [Zhang et al., 2020, Park et al., 2021], and the Quadratic Assignment Problem [Nowak et al., 2017].

A core challenge for deep learning approaches on CO is the lack of training data. Annotating such data requires the solution of a huge number of instances of the CO, hence such supervised learning approaches are computationally infeasible for NP-hard problems [Yehuda et al., 2020]. Circumventing this difficulty is key to unlocking the full potential of otherwise broadly applicable DNNs for CO.

Our work demonstrates how classical ideas in CO together with DNNs can lead to a scalable self-supervised learning approach, mitigating the lack of training data. Concretely, we focus on the Maximum Independent Set (MIS) problem: Given a graph $G(V, E)$, MIS asks for a set of nodes of maximum cardinality such that no two nodes in the selected set are connected with an edge. MIS is an NP-hard problem with several hand-crafted heuristics (e.g., *greedy heuristic*, *local search*). More recently, several deep learning approaches have been proposed [Karalias and Loukas, 2020, Toenshoff et al., 2019, Schuetz et al., 2022a]

Our approach involves the following steps to determine an MIS in a graph. We use graph neural networks (GNNs) [Wu et al., 2020] to enable a model to generate approximate maximum independent sets after training on data that was annotated by the model itself. For this purpose, we draw inspiration from dynamic programming (DP) and employ a DP-like recursive algorithm. Initially, we are given a graph. At each recursive step, we select a random vertex from that graph and create two sub-graphs: one by removing the selected vertex and another by removing all its neighboring vertices. We then

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

make a comparison between these sub-graphs to determine which sub-graph is likely to have a larger independent set, and we use the sub-graph with the highest estimated IS for the next recursive call. We repeat this process until we reach a graph consisting only of isolated vertices, which signifies the discovery of an independent set for the original graph.

Dynamic programming guarantees that if our predictions are accurate (i.e., we select the sub-graph with the largest MIS value), our recursive algorithm will always result in a maximum independent set. To make accurate predictions, we introduce "graph comparing functions," which take two graphs as input and output a winner. We implement such graph-comparing functions with GNNs.

We adopt a self-training approach to train our graph-comparing function and optimize the parameters of the GNN. In each epoch, we update the graph-comparing function parameters to ensure it accurately fits the data it has seen so far. The data comprises pairs of graphs (G, G') along with a label $\mathrm{Label}(G, G') \in \{0, 1\}$. For annotating the labels, we utilize the output of the recursive algorithm that leverages the graph-comparing function. Supported by theoretical and experimental evidence, we demonstrate how the self-annotation process improves parameter selection.

We conduct a thorough validation of our self-training approach in three real-world graph distribution datasets. Our algorithm surpasses the performance of previous deep learning methods [Karalias and Loukas, 2020, Toenshoff et al., 2019, Ahn et al., 2020] in the context of the MIS problem. To further validate the efficacy of our method, we explore its robustness on out-of-distribution data. Notably, our results demonstrate that the induced algorithm achieves competitive performance, showcasing the generalization capability of the learned comparator across different graph structures and distributions. In addition, we extend the evaluation of our DP-based self-training approach to tackle the Minimum Vertex Cover (MVC) problem in Appendix E. Encouragingly, similar to the MIS case, our induced GNN-based algorithms for MVC admit competitive performance with respect to other deep-learning approaches.

2 Related work

Our work lies in the intersection of various domains, i.e., combinatorial optimization, Dynamic Programming, and (graph) neural networks. We review the most critical ideas in each domain here and defer a more detailed discussion in Appendix A.

Graph Neural Networks (GNNs) have gained widespread popularity due to their ability to learn representations of graph-structured data [Xiao et al., 2022, Zhang and Chen, 2018, Zhu et al., 2021, Errica et al., 2019] invariant to the size of the graph. More complex architectural blocks, such as the Graph Convolutional Network (GCN) [Kipf and Welling, 2017, Zhang et al., 2019], the Graph Attention Network (GAT) [Veličković et al., 2017], and the Graph Isomorphism Network [Xu et al., 2018] have become influential instances of GNNs. In our work, we utilize a simple GNN architecture to showcase the effectiveness of our proposed framework. While our choice of architecture is intentionally simple, we emphasize its modular nature, which enables us to incorporate more complex GNNs with ease.

Combinatorial optimization: Supervised learning approaches have been used for tackling CO tasks, such as the Traveling Salesman Problem (TSP) [Vinyals et al., 2015], the Vehicle Routing Problem (VRP) [Shalaby et al., 2021], and Graph Coloring [Lemos et al., 2019]. Due to the graph structure of the problems, GNNs are often used for tackling those tasks [Prates et al., 2019, Nazari et al., 2018, Schuetz et al., 2022b]. However, owing to the computational overhead of obtaining supervised labels, such supervised approaches often do not scale well. Instead, unsupervised approaches have been deployed recently [Wang and Li, 2023]. A popular approach relies on a continuous relaxation of the loss function [Karalias and Loukas, 2020, Wang et al., 2022, Wang and Li, 2023]. In contrast to the previous unsupervised works, we adopt Dynamic Programming techniques to diminish the overall time complexity of the algorithm. Another approach uses reinforcement learning (RL) methods to address CO tasks, such as in Covering Salesman Problem [Li et al., 2021], the TSP [Zhang et al., 2022], the VRP [James et al., 2019], and the Minimum Vertex Cover (MVC) [Tian and Li, 2021]. However, applying RL to CO problems can be challenging because of the long learning time required and the non-differentiable nature of the loss function.

Dynamic Programming has been a powerful problem-solving technique since at least the 50s [Bellman, 1954]. In recent years, researchers have explored the use of deep neural networks (DNNs) to

replace the function responsible for dividing a problem into subproblems and estimating the optimal decision at each step [Yang et al., 2018]. Despite the progress, implementing CO tasks with Dynamic Programming suffers from significant computational overheads, since the size of the search space grows exponentially with the problem size [Xu et al., 2020]. Our approach overcomes this issue by utilizing a model that approximates the standard lookup table from Dynamic Programming, meaning that we avoid the exponential search space typically associated with DP.

3 An optimal solution to Maximum Independent Set (MIS)

Let us first introduce MIS and its relationship with Dynamic Programming.

Notation: $G(V, E)$ denotes an undirected graph where V stands for vertices and E for the edges. $\mathcal{N}(v)$ denotes the neighbors of vertex $v \in V, \mathcal{N}(v) = \{u \in V \ : \ (u, v) \in E\}$. The *degree* of vertex $v \in V$ is $d(v) := |\mathcal{N}(v)|$. Given a set of vertices $S \subseteq V$, G/S denotes the remaining graph of G after removing all nodes $v \in S$.

Definition 1 (Maximum Independent Set)**.** *Given an undirected graph $G(V, E)$, find a maximum set of nodes $S \subseteq V$ such that $(u, v) \notin E$ for all vertices $u, v \in S$. We denote with $|\mathrm{MIS}(G)|$ the size of the maximum independent set of graph G.*

Dynamic Programming is a powerful technique for algorithmic design in which the optimal solution of the instance of interest is constructed by combining the optimal solution of smaller sub-instances. The combination step is governed by local optimality conditions which, in the context of MIS, take the form of Theorem 1. Theorem 1 establishes that the decision to remove a node $v \in V$ or its neighbors $\mathcal{N}(v)$ during the recursive process depends on whether $|\mathrm{MIS}\left(G/\mathcal{N}(v)\right)| \geq |\mathrm{MIS}(G/v)\,|$. This decision continues until a graph with no edges is reached. According to Theorem 1, if at each step of the recursion, the choice is made based on whether $|\mathrm{MIS}\left(G/\mathcal{N}(v)\right)| \geq |\mathrm{MIS}(G/v)\,|$ or not, then the resulting empty graph is guaranteed to be an optimal solution. The proof is this theorem can be found in Appendix I.

Theorem 1. *Let a graph $G(V, E) \in \mathcal{G}$. Then for any vertex $v \in V$ with $d(v) \geq 1$,*

$$|\mathrm{MIS}(G)| = \max\left(|\mathrm{MIS}\left(G/\mathcal{N}(v)\right)|, |\mathrm{MIS}(G/\{v\})\,|\right)^{1}.$$

4 Graph Neural Network-based Algorithm for MIS

In this section, we present our approach for developing algorithms for MIS parameterized by parameters $\theta \in \Theta$. Initially in Sec. 4.1 we present how any *graph-comparing function* taking as input two different graphs and outputting a $\{0, 1\}$ value can be used in the construction of an algorithm computing an independent set (not necessarily optimal). In Sec. 4.2 we present how Graph Neural Networks can be used in the construction of such graph-comparing functions. Finally, in Sec. 4.3, we present our *inference algorithm* that computes an independent set of any graph $G \in \mathcal{G}$.

4.1 MIS Algorithms induced by Graph Comparing Functions

Consider a function $\mathrm{CMP} : \mathcal{G} \times \mathcal{G} \mapsto \{0, 1\}$ that compares two graphs G, G' based on the size of their MIS. Namely, if $|\mathrm{MIS}(G)| \geq |\mathrm{MIS}(G')|$ then $\mathrm{CMP}(G, G') = 0$ and $\mathrm{CMP}(G, G') = 1$ otherwise.

Theorem 1 ensures that if we have access in such a *graph-comparing function* then we can compute an independent set of maximum size for any graph G. From a starting node v, with an initial graph from G, and by recursively selecting either $G/\{v\}$ or $G/\mathcal{N}(v)$ based on $|\mathrm{MIS}(G/\{v\})| \geq |\mathrm{MIS}(G/\mathcal{N}(v))|$, we are ensured to end in an independent set of maximum size. The decision of whether $|\mathrm{MIS}(G/\{v\})| \geq |\mathrm{MIS}(G/\mathcal{N}(v))|$ at each recursive call can be made according to the output of $\mathrm{CMP}(G/\{v\}, G/\mathcal{N}(v))$.

The cornerstone idea of our approach is that *any graph-comparing function* CMP induces such a recursive algorithm for a MIS. Recursively selecting $G/\{v\}$ or $G/\mathcal{N}(v)$ based on the output of a graph generating function $\mathrm{CMP}(G/\{v\}, G/\mathcal{N}(v)) \in \{0, 1\}$ always guarantees to reach an

[1]Note: In the trivial case where G is an empty graph (i.e., it has no edges), the size of the maximum independent set is $|V|$.

Algorithm 1 Comparator-Induced Algorithm

function $\mathcal{A}^{\mathrm{CMP}}(G(V,E))$ ▷ Algorithm $\mathcal{A}^{\mathrm{CMP}}(G)$ takes a graph G as input
  if $|E| = 0$ **then return** V
  end if
  pick a vertex $v \in V$ with $d(v) > 0$ uniformly at random.
  $G_0 \leftarrow G \setminus \{v\}$ and $G_1 \leftarrow G \setminus \mathcal{N}(v)$
  if $\mathrm{CMP}(G_0, G_1) = 0$ **then**
    $G \leftarrow G_0$ ▷ Remove vertex v
  else
    $G \leftarrow G_1$ ▷ Remove the neighbors of v
  end if
  return $\mathcal{A}^{\mathrm{CMP}}(G)$
end function

independent set of the original graph. In case $\mathrm{CMP}(G, G') \neq \mathbb{I}\left[|\mathrm{MIS}(G)| < |\mathrm{MIS}(G')|\right]$, where $\mathbb{I}$ is the indicator function, it is not guaranteed that the computed independent set is of the maximum size. However, there might exist a reasonable graph comparing functions that $i)$ are efficiently computable $ii)$ lead to near-optimal solutions.

In Definition 2 and Algorithm 1 we formalize the idea above.

Definition 2. *A comparator* $\mathrm{CMP} : \mathcal{G} \times \mathcal{G} \mapsto \{0, 1\}$ *is a function taking as input two graphs* G, G' *and outputing a* $\{0, 1\}$ *value.*

Proposition 1. *Any comparator* $\mathrm{CMP} : \mathcal{G} \times \mathcal{G} \mapsto \{0, 1\}$ *induces a randomized algorithm* $\mathcal{A}^{\mathrm{CMP}}$ *(Algorithm 1).*

Remark 1. *Given a graph-comparing function* $\mathrm{CMP} : \mathcal{G} \times \mathcal{G} \mapsto \{0, 1\}$*, the induced algorithm is randomized, since at Step* 4 *of Algorithm 1, a vertex* v *is randomly selected. Notice that Algorithm 1 recursively proceeds until a subgraph with* 0 *edges is reached (see Step* 2*).*

Remark 2. *Two different comparators* CMP *and* CMP' *induce two difference algorithms* $\mathcal{A}^{\mathrm{CMP}}$ *and* $\mathcal{A}^{\mathrm{CMP}'}$ *for estimating the maximum independent set.*

4.2 Comparators through Graph Neural Networks

In this section, we discuss the architecture of a model $M_\theta : \mathcal{G} \mapsto \mathbb{R}$, parameterized by $\theta \in \Theta$, that is used for the construction of a comparator function

$$\mathrm{CMP}_\theta(G, G') = \mathbb{I}\left[M_\theta(G) < M_\theta(G')\right] \ .$$

In order to embed graph-level information, we introduce a new GNN module, which we refer to as the Graph Embedding Module (GEM). Unlike standard GNN modules, this module captures different semantic meanings of differing embeddings of a node, its neighbors, and anti-neighbors.

Graph Embedding Module (GEM):

The GEM operates using the following recursive formula:

$$\mu_v^{k+1} = \mathrm{LN}\left(\mathrm{GELU}\left(\left[\theta_0^k \mu_v^k \middle\| \theta_1^k \sum_{u \in \mathcal{N}(v)} \mu_u^k \middle\| \theta_2^k \sum_{u \notin \mathcal{N}(v)} \mu_u^k\right]\right)\right) \ . \tag{1}$$

Initially, all nodes in this graph have zeros embeddings $\mu_v^0 = \vec{0} \in \mathbb{R}^{3p}$. Here, μ_v^0 denotes the initial embedding vector of node v. In Eq. (1), for all iterations $k \in [0, \ldots, K-1]$, the embeddings of a node denoted by $\mu_v^k \in \mathbb{R}^{3p}$, its neighbors, and its anti-neighbors v are put through their own linear layers, denoted by $\theta_0^k, \theta_1^k, \theta_2^k \in \mathbb{R}^{p \times 3p}$, which are the parameters of the module. The bias term is omitted in the equation for readability purposes. We incorporate anti-neighbors in the GEM to capture complementary relationships between nodes. By using separate linear layers for different

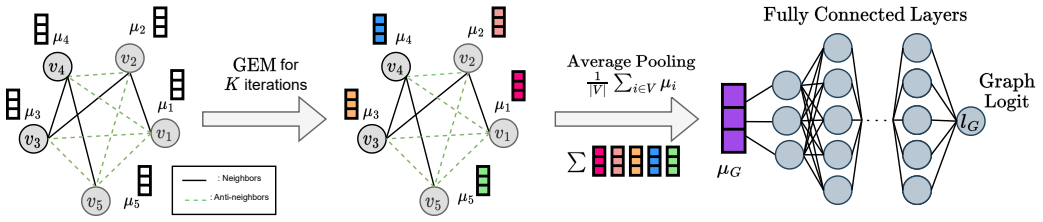

Figure 1: Architecture of model $M_\theta(G)$. From left to right: initially, an input graph G is passed into the model with zeros as node embeddings, which are displayed as white in the figure. The striped green edges connect the anti-neighbors, which are also used in the GEM. After K iterations of the GEM module, the final node embeddings are obtained. These are then averaged to obtain a graph embedding μ_G. Finally, the graph embedding is put through multiple fully-connected layers to obtain a final logit value for the input graph.

features, we emphasize the contrasting semantic meaning between neighbors and anti-neighbors, representing negative and positive relationships in the graph. Then, the individual feature embeddings are concatenated, which is denoted by $[\dots \| \dots]$, followed by a GELU activation function [Hendrycks and Gimpel, 2016] and layer normalization [Ba et al., 2016a].

The complete model architecture is depicted in Fig. 1. At a high level, the architecture uses a Graph Embedding Module to extract a global graph embedding from the input graph, which is then passed through a set of fully connected layers to output a logit for that graph. During the training process of the comparator function, we utilize $\mathrm{CMP}_\theta(G, G') = \mathrm{softmax}\left([M_\theta(G) \| M_\theta(G')]\right)$, which forms a differentiable loss function for classification.

4.3 Inference Algorithm

In the previous section, we discussed how a parameterization $\theta \in \Theta$ defines the graph-comparing function $\mathrm{CMP}_\theta(G, G') = \mathbb{I}\left[M_\theta(G) < M_\theta(G')\right]$. As a result, the same parameterization $\theta \in \Theta$ defines an algorithm $\mathcal{A}^{\mathrm{CMP}_\theta}$, where at Step 6 of Algorithm 1, the comparing function CMP_θ is used.

5 Self-Supervised Training through the Consistency Property

In this section, we present our methodology for selecting the parameters $\theta \in \Theta$ so that the resulting inference algorithm $\mathcal{A}^{\mathrm{CMP}_\theta}(\cdot)$ computes independent sets with (close to) the maximum value.

The most straightforward approach is to select the parameters $\theta \in \Theta$ such that $\mathrm{CMP}_\theta(G, G') \simeq \mathbb{I}\left[|\mathrm{MIS}(G)| < |\mathrm{MIS}(G')|\right]$ using labeled data. The problem with this approach is that a huge amount of annotated data of the form $\{((G, G'), \mathbb{I}\left[|\mathrm{MIS}(G)| < |\mathrm{MIS}(G')|\right])\}$ are required. Since finding the MIS is an NP-Hard problem, annotating such data comes with an insurmountable computational burden.

The **key idea** to overcome the latter limitation is to annotate the data of the form $\{(G, G')\}$ by using the algorithm $\mathcal{A}^{\mathrm{CMP}_\theta}(\cdot)$ that runs in polynomial time with respect to the size of the graph. Intuitively, our proposed framework entails the optimization of the parameterized comparator function CMP_θ on data generated using algorithm $\mathcal{A}^{\mathrm{CMP}_\theta}$. A better comparator function leads to a better algorithm, which leads to better data, and vice versa. This mutually reinforcing relationship between the two components of our framework is theoretically indicated by Theorem 2 that we present in Section 5.1. The exact steps are detailed below.

5.1 Consistent Graph Comparing Functions

In this section, we introduce the notion of a *consistent* graph-comparing function (Definition 3) that plays a critical role in our self-supervised learning approach. Kindly take note that $\mathcal{A}^{\mathrm{CMP}}$ utilizes the unparameterized variant of a comparator function, whereas $\mathcal{A}^{\mathrm{CMP}_\theta}$ utilizes its parameterized counterpart.

Algorithm 2 Basic Pipeline of our Training Approach

Input: A distribution $\mathcal{D}$ over graphs.
Initialize parameters $\theta_0 \in \Theta$.
Initialize a *graph-buffer* $\mathcal{B} \leftarrow \varnothing$.
for each epoch $t = 0, \ldots, T-1$ **do**
    Sample a graph $G_{\text{init}} \sim \mathcal{D}$.
    Run $\mathcal{A}^{\text{CMP}_{\theta_t}}(G)$ and store in $\mathcal{B}$ all graphs produced during each recursive call of Algorithm 1.
    Update the parameters $\theta_{t+1} \in \Theta$ such that

$$\theta^{t+1} := \text{argmin}_{\theta \in \Theta} \mathbb{E}_{(G,G') \sim \mathcal{B}} \left[\ell \left(\text{CMP}_{\theta}(G, G'), \mathbb{I} \left[\mathbb{E} \left[|\mathcal{A}^{\text{CMP}_{\theta_t}}(G)| \right] < \mathbb{E} \left[|\mathcal{A}^{\text{CMP}_{\theta_t}}(G')| \right] \right] \right) \right]$$

end for

Definition 3 (Consistency)**.** *A graph-comparing function* $\text{CMP} : \mathcal{G} \times \mathcal{G} \mapsto \{0, 1\}$ *is called consistent if and only if for any pair of graphs* $G, G' \in \mathcal{G}$*,*

$$\text{CMP}(G, G') = 0 \text{ if and only if } \mathbb{E}\left[\left|\mathcal{A}^{\text{CMP}}(G)\right|\right] \geq \mathbb{E}\left[\left|\mathcal{A}^{\text{CMP}}(G')\right|\right].$$

Remark 3. *In Definition 3 we use* $\mathbb{E}\left[\left|\mathcal{A}^{\text{CMP}}(G)\right|\right], \mathbb{E}\left[\left|\mathcal{A}^{\text{CMP}}(G')\right|\right]$ *since, as we have already discussed, a comparator* CMP *induces a randomized algorithm* $\mathcal{A}^{\text{CMP}}$*.*

In Theorem 2, we formally establish that any *consistent graph-comparing function* CMP induces an optimal algorithm for the MIS.

Theorem 2. *Let a consistent comparator* $\text{CMP} : \mathcal{G} \times \mathcal{G} \mapsto \{0, 1\}$*. Then the algorithm* $\mathcal{A}^{\text{CMP}}(\cdot)$ *always computes a Maximum Independent Set,* $\mathbb{E}\left[\left|\mathcal{A}^{\text{CMP}}(G)\right|\right] = |\text{MIS}(G)|$ *for all* $G \in \mathcal{G}$*.*

Theorem 2 guarantees that any consistent graph comparing function CMP induces an optimal algorithm $\mathcal{A}^{\text{CMP}}$ for MIS. The proof for this theorem can be found in Appendix I. Hence, the selection of parameters $\theta^\star \in \Theta$ should be selected such that $\text{CMP}_{\theta^\star}$ is *consistent*. More precisely:

Goal of Training: Find parameters $\theta^\star \in \Theta$ such that for all $G, G' \in \mathcal{G}$:

$$\text{CMP}_{\theta^\star}(G, G') = 0 \text{ if and only if } \mathbb{E}\left[\left|\mathcal{A}^{\text{CMP}_{\theta^\star}}(G)\right|\right] \geq \mathbb{E}\left[\left|\mathcal{A}^{\text{CMP}_{\theta^\star}}(G')\right|\right].$$

5.2 Training a Consistent Comparator

The cornerstone idea of our self-supervised learning approach is to make the comparator more and more consistent over time. Namely, the idea is to update the parameters as follows:

$$\theta^{t+1} := \text{argmin}_{\theta \in \Theta} \mathbb{E}_{G,G'} \left[\ell \left(\text{CMP}_{\theta}(G, G'), \mathbb{I} \left[\mathbb{E} \left[\left|\mathcal{A}^{\text{CMP}_{\theta_t}}(G)\right| \right] < \mathbb{E} \left[\left|\mathcal{A}^{\text{CMP}_{\theta_t}}(G')\right| \right] \right] \right) \right] \quad (2)$$

where $\ell(\cdot, \cdot)$ is a binary classification loss. In Eq. (2), θ_t are the fixed parameters of the previous epoch. Thus, in the next few paragraphs, we only use the notation $\mathcal{A}^{\text{CMP}_{\theta_t}}$ to denote the fixed parameters. Gradient updates are only computed over θ.

Remark 4. *We remark that neither solving the non-convex minimization problem of Eq.* (2) *nor the existence of parameters* $\theta^\star \in \Theta$ *such that* $\text{CMP}_{\theta^\star}$ *can be guaranteed. However, using a first-order method for Eq.* (2) *and a large enough parameterization can lead to an approximately consistent comparator with approximately optimal performance.*

In Algorithm 2, we present the basic pipeline of the self-training approach that selects the parameters $\theta \in \Theta$ such that the inference algorithm $\mathcal{A}^{\text{CMP}_{\theta_t}}$ admits a competitive performance given as input graphs G following a graph-distribution $\mathcal{D} \subseteq \mathcal{G}$. However, while the basic pipeline of our self-training approach follows Algorithm 2, there are several differences and tweaks that we incorporate into our training process.

Creating the graph buffer $\mathcal{B}$**:** We are given a shuffled dataset of graphs $\mathcal{D}$, which represents the training data for the model. The core difference between the pipeline and the training process

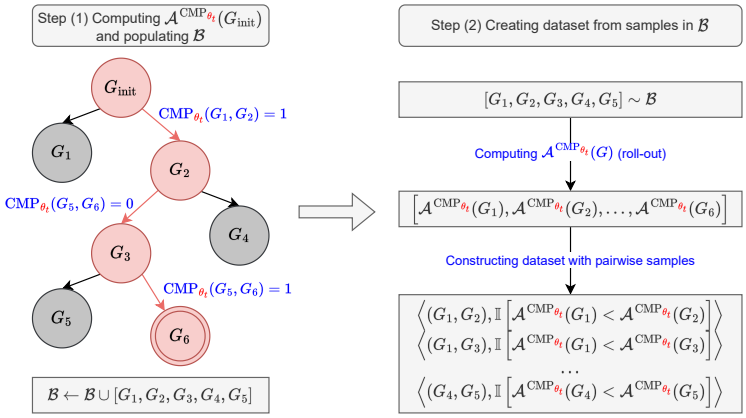

Figure 2: An example of data generation for the training. (Left) At the beginning of each training epoch (Step 5), Algorithm 2 samples $G_{\text{init}} \sim \mathcal{D}$ and computes an independent set by using the comparator CMP_{θ_t} and by following the branches in the *recursion tree* that are marked red, with the doubly circled one being the produced independent set. The generated graphs from this procedure are added to the buffer $\mathcal{B}$. (Right) Then, a dataset is created by sampling graphs from the buffer and then computing an estimate of their MIS size (based on $\mathcal{A}^{\mathrm{CMP}_{\theta_t}}$). Based on this estimate, a dataset is created with graph pairs (G, G') and their corresponding binary labels denoting which MIS estimates are larger.

comes from the graph buffer $\mathcal{B}$. In Algorithm 2, this buffer stores any graph G that is found during the recursive call of $\mathcal{A}^{\mathrm{CMP}_{\theta_t}}(G_{\text{init}})$ on $G_{\text{init}} \sim D$ (Step 6 of Algorithm 2). However, in the implementation of the graph buffer, it stores pairs of graphs (G, G') that were generated by $\mathcal{A}^{\mathrm{CMP}_{\theta_t}}(G_{\text{init}})$, alongside a binary label that indicates which of the two graphs has a larger estimated MIS size. How this estimate is generated, will be explained further down this section.

The training process: Prior to starting training, we first set two hyperparameters: one that specifies the number of graphs used to populate the buffer before training the model, and another that determines the number of graph pairs generated from $\mathcal{A}^{\mathrm{CMP}_{\theta_t}}(G)$ per graph $G \sim \mathcal{D}$. Then, a dataset is created by generating these pairs for the set number of graphs. The dataset is then added to the graph buffer, replacing steps 4, 5, and 6 in Algorithm 2, which only does this with one graph per epoch. Next, training starts, and after completing a set number of epochs, a new dataset is created using the updated model, and the process is repeated iteratively.

Estimating the MIS: Finally, the loss function in Step 7 of Algorithm 2, also operates slightly differently. The main difference arises from $\mathbb{I}\left[\mathbb{E}\left[|\mathcal{A}^{\mathrm{CMP}_{\theta_t}}(G)|\right] < \mathbb{E}\left[|\mathcal{A}^{\mathrm{CMP}_{\theta_t}}(G')|\right]\right]$, since the estimates $|\mathrm{MIS}(G)| \approx \mathbb{E}\left[|\mathcal{A}^{\mathrm{CMP}_{\theta_t}}(G)|\right]$ and $|\mathrm{MIS}(G')| \approx \mathbb{E}\left[|\mathcal{A}^{\mathrm{CMP}_{\theta_t}}(G')|\right]$ are not directly utilized. Instead, we propose two other approaches, which are better approximations than the expectations used in Algorithm 2, since they use a maximizing operator.

The first approach involves performing so-called "roll-outs" on the graph pairs generated G and G' by $\mathcal{A}^{\mathrm{CMP}_{\theta_t}}$, in order to estimate their MIS sizes. To perform the roll-outs, we simply run $\mathcal{A}^{\mathrm{CMP}_{\theta_t}}$ on graphs G and G' m times and use the maximum size of the found independent sets as an estimate of their MIS. Formally, in a roll-out on a graph G, we sample the independent sets $\mathrm{ISS}_1, \mathrm{ISS}_2, \ldots, \mathrm{ISS}_m \sim \mathcal{A}^{\mathrm{CMP}_{\theta_t}}(G)$. Then, the estimate of the MIS size of G is $\max\left(|\mathrm{ISS}_1|, |\mathrm{ISS}_2|, \ldots, |\mathrm{ISS}_m|\right)$.

An example of the entire process of generating the dataset using roll-outs can be found in Fig. 2.

Mixed roll-out variant: We introduce a variant of the aforementioned method, which utilizes the deterministic greedy algorithm. This greedy algorithm iteratively creates an independent set by removing the node with the lowest degree and adding it to the independent set. This algorithm is often an efficient approximation to the optimum solution. Our variant is constructed as follows: we compute the maximum between the roll-outs of the model and the result of the greedy algorithm, which creates a dataset with more accurate self-supervised approximations of the MIS values. This, in turn, generates binary targets for the buffer that are more likely to be accurate. Thus, for this second variant, the estimate of the MIS size of a graph G would be $\max\left(|\mathrm{Greedy}(G)|, |\mathrm{ISS}_1|, |\mathrm{ISS}_2|, \ldots, |\mathrm{ISS}_m|\right)$.

6 Experiments

In this section, we conduct an evaluation of the proposed method for the MIS problem. Let us first describe the training setup, the baselines, and the datasets. Additional details and experiments on MIS are displayed in the Appendices C and D. Our method also generalizes well in MVC, as the results in Appendix E illustrate.

6.1 Training setup

Our model: We implement two comparator models: one using just roll-outs with the model, and another using the roll-outs together with greedy, called "mixed roll-out". We train each model using a graph embedding module with $K = 3$ iterations, which takes in 32-dimensional initial node embeddings.

Baselines: We compare against the neural approaches *Erdos GNN* [Karalias and Loukas, 2020], *RUN-CSP* from Toenshoff et al. [2019], and a method specifically for the MIS problem: LwDMIS [Ahn et al., 2020]. Since we observe unexpected performance from RUN-CSP on the COLLAB and RB datasets, we have omitted those results from the table. We train every model for 300 epochs. Each experiment is performed on a single GPU with 6GB RAM.

Besides neural approaches, we use traditional baselines, such as the *Greedy MIS* [Wormald, 1995] , *Simple Local Search* [Feo et al., 1994] and a *Random Comparator* as a sanity check. Furthermore, we implement two mixed-integer linear programming solvers: *SCIP 8.0.3* and the highly optimized commercial solver *Gurobi 10.0*.

Datasets: We evaluate our model on three standard datasets, following Karalias and Loukas [2020]: COLLAB [Yanardag and Vishwanathan, 2015], TWITTER [Leskovec and Krevl, 2014] and RB [Xu et al., 2007, Toenshoff et al., 2019]. In addition, we introduce the SPECIAL dataset that includes challenging graphs for handcrafted approaches as we detail in Appendix C.

6.2 Results

Table 1 reports the average approximation ratios on the test instances of the various datasets. The approximation ratio is computed by dividing a solution's independent set size by the optimum solution, which is computed using the Gurobi solver with a time limit of 1 hour per graph.

The results indicate that the greedy algorithm performs strongly in three of the four datasets, which is consistent with the observation of Angelini and Ricci-Tersenghi [2022]. However, notice that our proposed approach outperforms the greedy in both the Twitter and the SPECIAL datasets, which validates that the greedy heuristic is not optimal in every case and is prone to failing in few cases. Importantly, among the neural approaches that are the main compared methods, our proposed method performs favorably in all datasets. The performance of our method indicates that the proposed self-training scheme is able to learn from diverse data distributions and generalize reasonably well in the test sets of the respective dataset. In addition, the proposed method is faster than the rest neural approaches.

The mixed roll-out model in Table 1 outperforms the normal roll-out model in almost all datasets, indicating the effectiveness of the greedy heuristic in roll-outs. This is particularly evident in the RB dataset. However, for SPECIAL instances, the normal model performs marginally better, possibly due to the unsuitability of the greedy guiding heuristic as a baseline for this dataset.

Out of distribution : We examine the performance of the learned comparator through its generalization to new graph distributions. Concretely, we conduct an out-of-distribution analysis as follows: each model is trained in one graph distribution, indicated by the rows of Table 2. Then, the model is evaluated on different graph distributions, indicated by the columns of Table 2. The analysis is conducted on both our model and the approach of *Erdos GNN* [Karalias and Loukas, 2020].

Surprisingly, our model trained over COLLAB displays good generalization skills across different datasets, even outperforming the RB-trained model on the RB dataset. Conversely, *Erdos GNN* trained over RB performs poorly over the COLLAB dataset. Both models trained over the RB dataset perform more poorly in general, likely due to the highly specific graph distribution of the RB dataset. Moreover, our model, on the whole, exhibits good generalization skills over different graph distributions.

Table 1: Test set approximation ratios (higher is better; the best performance in bold) on four datasets. We report the average approximation ratios (along with std and time budget) on MIS. Notice that the proposed method outperforms all the deep-learning-based approaches across datasets.

Method (↓) Dataset (→)	RB	COLLAB	TWITTER	SPECIAL
CMP (Normal Roll-outs)	0.770 ± 0.107 (0.43 s/g)	0.990 ± 0.051 (0.17 s/g)	0.967 ± 0.083 (0.35 s/g)	$\mathbf{0.996 \pm 0.029}$ (0.04 s/g)
CMP (Mixed Roll-outs)	$\mathbf{0.836 \pm 0.083}$ (0.36 s/g)	$\mathbf{0.990 \pm 0.049}$ (0.21 s/g)	$\mathbf{0.977 \pm 0.031}$ (0.21 s/g)	0.994 ± 0.035 (0.05 s/g)
Erdos' GNN	0.813 ± 0.107 (1.39 s/g)	0.952 ± 0.142 (0.60 s/g)	0.935 ± 0.078 (1.37 s/g)	0.921 ± 0.218 (1.03 s/g)
LwDMIS	0.804 ± 0.089 (0.42 s/g)	0.978 ± 0.031 (0.17 s/g)	0.972 ± 0.032 (0.19 s/g)	0.828 ± 0.304 (0.32 s/g)
RUN-CSP (Accurate)	–	–	0.875 ± 0.053 (0.57 s/g)	0.946 ± 0.059 (0.51 s/g)
Greedy MIS	0.925 ± 0.053 (0.01 s/g)	0.998 ± 0.023 (0.02 s/g)	0.964 ± 0.048 (0.04 s/g)	0.131 ± 0.055 (0.03 s/g)
Random CMP	0.615 ± 0.155 (0.42 s/g)	0.817 ± 0.211 (0.30 s/g)	0.634 ± 0.182 (0.36 s/g)	0.225 ± 0.279 (0.41 s/g)
Simple Local Search (10s)	0.565 ± 0.237	0.860 ± 0.213	0.644 ± 0.218	0.188 ± 0.340
SCIP 8.0.3 (1s)	0.741 ± 0.351	0.999 ± 0.016	0.959 ± 0.024	1.000
SCIP 8.0.3 (5s)	0.937 ± 0.118	1.000	0.999 ± 0.024	1.000
Gurobi 10.0 (0.5s)	0.969 ± 0.070	0.981 ± 0.068	0.985 ± 0.085	0.940 ± 0.237
Gurobi 10.0 (1s)	0.983 ± 0.051	1.000	1.000	1.000
Gurobi 10.0 (5s)	0.999 ± 0.008	1.000	1.000	1.000

Table 2: Out-of-distribution approximation ratios during inference (higher is better). Every row denotes a model trained on a specific dataset. Every column considers a different test dataset. The CMP is trained using mixed roll-outs. Notice that the proposed method generalizes well in out-of-distribution structures. This is indicative of the learned comparator extracting robust patterns.

Model (↓) Dataset (→)	RB	COLLAB	TWITTER
CMP RB	–	0.903 ± 0.186	0.668 ± 0.187
CMP COLLAB	0.856 ± 0.080	–	0.906 ± 0.094
CMP TWITTER	0.773 ± 0.101	0.927 ± 0.148	–
Erdos' GNN RB	–	0.361 ± 0.334	0.752 ± 0.188
Erdos' GNN COLLAB	0.680 ± 0.071	–	0.592 ± 0.186
Erdos' GNN TWITTER	0.746 ± 0.092	0.666 ± 0.385	–

7 Conclusion

Motivated by the principles of Dynamic Programming, we develop a self-training approach for important CO problems, such as the Maximum Independent Set and the Minimum Vertex Cover. Our approach embraces the power of self-training, offering the dual benefits of data self-annotation and data generation. These inherent attributes are instrumental in providing an unlimited source of data indicating that the performance of the induced algorithms can be significantly improved with sufficient scaling on the computational resources. We firmly believe that a thorough investigation into the interplay between Dynamic Programming and self-training techniques can pave the way for new deep-learning-oriented approaches for demanding CO problems.

Limitations: Our current empirical approach lacks theoretical guarantees on the convergence or the approximate optimality of the obtained algorithm. Additionally, the implemented GNN is using core modules, while more complex modules could result in further empirical improvements, which can be the next step in this direction.

References

Sungsoo Ahn, Younggyo Seo, and Jinwoo Shin. Learning what to defer for maximum independent sets. *International Conference on Machine Learning (ICML)*, 2020.

Maria Chiara Angelini and Federico Ricci-Tersenghi. Cracking nuts with a sledgehammer: when modern graph neural networks do worse than classical greedy algorithms. *arXiv preprint arXiv:2206.13211*, 2022.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016a.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016b.

Richard Bellman. The theory of dynamic programming. *BULL*, 60(6):503–515, 1954.

Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *JACM*, 9(1): 61–63, 1962.

John R Current and David A Schilling. The covering salesman problem. *Transportation science*, 23 (3), 1989.

Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. *arXiv preprint arXiv:1912.09893*, 2019.

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

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *NIPS*, 32, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Yujiao Hu, Yuan Yao, and Wee Sun Lee. A reinforcement learning approach for optimizing multiple traveling salesman problems over graphs. *Knowledge-Based Systems*, 204:106244, 2020.

JQ James, Wen Yu, and Jiatao Gu. Online vehicle routing with neural combinatorial optimization and deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 20 (10):3806–3817, 2019.

Steven James, George Konidaris, and Benjamin Rosman. An analysis of monte carlo tree search. *AAAI Conference on Artificial Intelligence*, 31(1), Feb. 2017. URL `https://ojs.aaai.org/index.php/AAAI/article/view/11028`.

Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. *Advances in neural information processing systems (NeurIPS)*, 33:6659–6672, 2020.

Maryam Karimi-Mamaghan, Mehrdad Mohammadi, Patrick Meyer, Amir Mohammad Karimi-Mamaghan, and El-Ghazali Talbi. Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. *EJOR*, 296(2):393–422, 2022.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations (ICLR)*, 2017.

Henrique Lemos, Marcelo Prates, Pedro Avelar, and Luis Lamb. Graph colouring meets deep learning: Effective graph neural network models for combinatorial problems. In *International Conference on Tools for Artificial Intelligence (ICTAI)*, pages 879–885. IEEE, 2019.

Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

Kaiwen Li, Tao Zhang, Rui Wang, Yuheng Wang, Yi Han, and Ling Wang. Deep reinforcement learning for combinatorial optimization: Covering salesman problems. *IEEE Transactions on Cybernetics*, 52(12):13142–13155, 2021.

Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. *Advances in neural information processing systems (NeurIPS)*, 31, 2018.

Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *COR*, 134:105400, 2021.

Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *International Conference on Machine Learning (ICML)*, 2020. URL `www.graphlearning.io`.

Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. *Advances in neural information processing systems (NeurIPS)*, 31, 2018.

Alex W. Nowak, Soledad Villar, Afonso S. Bandeira, and Joan Bruna. A note on learning algorithms for quadratic assignment with graph neural networks. *ArXiv*, abs/1706.07450, 2017.

Junyoung Park, Sanjar Bakhtiyar, and Jinkyoo Park. Schedulenet: Learn to solve multi-agent scheduling problems with reinforcement learning, 2021.

Marcelo Prates, Pedro HC Avelar, Henrique Lemos, Luis C Lamb, and Moshe Y Vardi. Learning to solve np-complete problems: A graph neural network for decision tsp. In *AAAI Conference on Artificial Intelligence*, volume 33, pages 4731–4738, 2019.

Martin JA Schuetz, J Kyle Brubaker, and Helmut G Katzgraber. Combinatorial optimization with physics-inspired graph neural networks. *Nature Machine Intelligence*, 4(4):367–377, 2022a.

Martin JA Schuetz, J Kyle Brubaker, Zhihuai Zhu, and Helmut G Katzgraber. Graph coloring with physics-inspired graph neural networks. *Physical Review Research*, 4(4):043131, 2022b.

Ulrich Schwalbe and Paul Walker. Zermelo and the early history of game theory. *Games and economic behavior*, 34(1):123–137, 2001.

Mohamed A Wahby Shalaby, Ayman R Mohammed, and Sally S Kassem. Supervised fuzzy c-means techniques to solve the capacitated vehicle routing problem. *IAJIT*, 18(3A):452–463, 2021.

Hong Tian and Dazi Li. Graph convolutional neural networks with am-actor-critic for minimum vertex cover problem. In *2021 33rd Chinese Control and Decision Conference (CCDC)*, pages 3841–3846. IEEE, 2021.

Jan Toenshoff, Martin Ritzert, Hinrikus Wolf, and Martin Grohe. Run-csp: unsupervised learning of message passing networks for binary constraint satisfaction problems. *CoRR*, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems (NeurIPS)*, pages 5998–6008, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *NeurIPS*, 28, 2015.

Haoyu Wang and Pan Li. Unsupervised learning for combinatorial optimization needs meta-learning. *arXiv preprint arXiv:2301.03116*, 2023.

Haoyu Peter Wang, Nan Wu, Hang Yang, Cong Hao, and Pan Li. Unsupervised learning for combinatorial optimization with principled objective relaxation. In *NIPS*, 2022.

Nicholas C Wormald. Differential equations for random processes and random graphs. *AAP*, pages 1217–1235, 1995.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

Shunxin Xiao, Shiping Wang, Yuanfei Dai, and Wenzhong Guo. Graph neural networks in node classification: survey and evaluation. *Mach. Vis. Appl.*, 33:1–19, 2022.

Zhihao Xing and Shikui Tu. A graph neural network assisted monte carlo tree search approach to traveling salesman problem. *IEEE Access*, 8:108418–108428, 2020.

Zhihao Xing, Shikui Tu, and Lei Xu. Solve traveling salesman problem by monte carlo tree search and deep neural network. *arXiv preprint arXiv:2005.06879*, 2020.

Ke Xu, Frédéric Boussemart, Fred Hemery, and Christophe Lecoutre. Random constraint satisfaction: Easy generation of hard (satisfiable) instances. *Artificial intelligence*, 171(8-9):514–534, 2007.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Ruiyang Xu and Karl Lieberherr. Learning self-play agents for combinatorial optimization problems. *KER*, 35:e11, 2020.

Shenghe Xu, Shivendra S Panwar, Murali Kodialam, and TV Lakshman. Deep neural network approximated dynamic programming for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1684–1691, 2020.

Pinar Yanardag and S. V. N. Vishwanathan. Deep graph kernels. *SIGKDD*, 2015.

Feidiao Yang, Tiancheng Jin, Tie-Yan Liu, Xiaoming Sun, and Jialin Zhang. Boosting dynamic programming with neural networks for solving np-hard problems. In *ACML*, pages 726–739. PMLR, 2018.

Gal Yehuda, Moshe Gabel, and Assaf Schuster. It's not what machines can learn, it's what we cannot teach. In *ICML*, pages 10831–10841. PMLR, 2020.

Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Xu Chi. Learning to dispatch for job shop scheduling via deep reinforcement learning. *Advances in neural information processing systems (NeurIPS)*, 33:1621–1632, 2020.

Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *NeurIPS*, 31, 2018.

Rongkai Zhang, Cong Zhang, Zhiguang Cao, Wen Song, Puay Siew Tan, Jie Zhang, Bihan Wen, and Justin Dauwels. Learning to solve multiple-tsp with time window and rejections via deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 2022.

Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 6(1):1–23, 2019.

Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. Neural bellman-ford networks: A general graph neural network framework for link prediction. *NeurIPS*, 34:29476–29490, 2021.