Deep Attentive Belief Propagation: Integrating Reasoning and Learning for Solving Constraint Optimization Problems

Anonymous Author(s)
Affiliation
Address
email

Abstract

Belief Propagation (BP) is an important message-passing algorithm for various reasoning tasks over graphical models, including solving the Constraint Optimization Problems (COPs). It has been shown that BP can achieve state-of-the-art performance on various benchmarks by mixing old and new messages before sending the new one, i.e., *damping*. However, existing methods on tuning a *static* damping factor for BP not only is laborious but also harms their performance. Moreover, existing BP algorithms treat each variable node's neighbors equally when composing a new message, which also limits their exploration ability. To address these issues, we seamlessly integrate BP, Gated Recurrent Units (GRUs), and Graph Attention Networks (GATs) within the massage-passing framework to reason about *dynamic* weights and damping factors for composing new BP messages. Our model, Deep Attentive Belief Propagation (DABP), takes the factor graph and the BP messages in each iteration as the input and infers the optimal weights and damping factors through GRUs and GATs, followed by a multi-head attention layer. Furthermore, unlike existing neural-based BP variants, we propose a novel *self-supervised* learning algorithm for DABP with a smoothed solution cost, which does not require expensive training labels and also avoids the common out-of-distribution issue through efficient online learning. Extensive experiments show that our model significantly outperforms state-of-the-art baselines.

1 Introduction

Belief Propagation (BP) [20, 28] is an important message-passing algorithm for various reasoning tasks over graphical models, e.g., computing partition function of a Markov random field [40], estimating the marginal distribution of a given set of random variables [1], and decoding LPDC codes [24]. Constraint Optimization Problems (COPs) [25, 33] are a general mathematical paradigm for modeling many real-world problems like transportation, supply chain, energy, finance, and scheduling [5, 33]. When solving COPs, BP, also known as Min-sum message passing [14], seeks to find a cost-optimal solution by propagating cost information over the corresponding factor graph.

It is known that vanilla BP does not guarantee convergence on a factor graph containing loops and thus, it often explores low-quality solutions on COPs with cyclic factor graphs due to excessive loopy propagation. Therefore, considerable research efforts [6, 8, 30, 31, 32, 44] have been devoted

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

to tackling the convergence issue of loopy BP. Among them, Damped BP (DBP)[1] [8] has drawn significant attention recently. By mixing the new message composed in each iteration with the old message composed in the previous iteration, i.e., damping, it has been shown that DBP with a suitable damping factor will often converge and achieve state-of-the-art performance in practice. In fact, damping can be considered as a degree of freedom to balance exploration and exploitation in BP [8].

Due to the significant impact of damping on BP [43], a damping factor is often treated as a hyperparameter and requires laborious case-by-case tuning. Besides, existing works often use a homogeneous and static damping factor for all variable-nodes which limits DBP's capability. Moreover, DBP simply treats each variable-node's neighbors equally when composing a new message, which fails to explore optimal composition strategy. On the other hand, Deep Neural Networks (DNNs) have been applied to boost the performance of BP [21, 35, 41]. However, existing DNN-based BP variants often require supervised learning and expensive training labels and thus, they cannot be directly applied to solve COPs when training labels are not available. They also manipulate or simulate the BP messages with some black-box DNN components, which destroys the semantic of a BP message being the weighted sum of cost functions' marginalizations and thus, the variable decision-making rule of BP for COPs is no longer applied (i.e., selecting the assignments with the minimum beliefs).

To address the above issues, we propose the *first self-supervised* DNN-based BP for solving COPs by seamlessly integrating BP, Gated Recurrent Units (GRUs) [7], and Graph Attention Networks (GATs) [38] within the massage-passing framework to reason about *dynamic* weights and damping factors for composing new BP messages. Specifically, we make the following key contributions:

- We extend DBP by allowing dynamic damping factors and different neighbor weights for each variable-node. Consequently, we have more fine-grained control for composing new messages to trigger effective exploration without altering the semantic of BP messages.
- We infer the optimal damping factors and neighbor weights for each iteration automatically by seamlessly integrating BP and DNNs. Our model, Deep Attentive Belief Propagation (DABP), firstly embeds respectively the factor graph and BP messages with GATs and GRUs, and infers damping factors and neighbor weights through a multi-head attention [2, 37] layer; Then, BP runs with those updated factors and weights to compose and propagate new messages, and so on. Note that integrating GRUs can capture the dynamics of BP messages and avoid the gradient vanishing/exploding issue in long BP reasoning sequences.
- We further equip our DABP with a novel self-supervised learning loss which is a smoothed surrogate for the cost under BP decision-making rule. Therefore, unlike existing DNN-based BP variants, we do not require expensive training labels which are often not available in practice, and also our model can avoid out-of-distribution issue with efficient online learning.
- We conduct extensive experimental evaluations on four standard benchmarks. The results show that our DABP achieves significantly higher convergence rate, where it successfully converges on average 96.25% instances given 1000 iterations, and it also outperforms the state-of-the-art baselines by considerable margins.

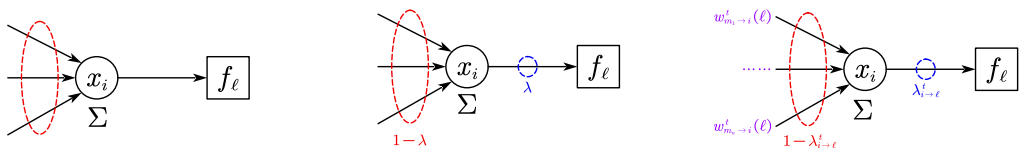
2 Backgrounds

In this section, we review the backgrounds of COPs, factor graphs, and Min-sum BP.

2.1 Constraint Optimization Problems and Factor Graph

A Constraint Optimization Problem (COP) [25] is defined by a triplet $\langle X, D, F \rangle$, which corresponds to the set of variables, domains, and constraint functions, respectively. Each variable $x_i \in X$ is associated with a finite domain $D_i \in D$. Each constraint function $f_\ell \in F$ with scope $scp(f_\ell) \subseteq X$ specifies the cost for each possible assignment of the variables in its scope. The objective is to find a

[1] The method is referred as "Damped Max-sum" in [8]. We use Damped BP for a coherent presentation.



(a) Vanilla BP (b) DBP (c) DABP

Figure 1: Comparison of BP variants. Obviously, our DABP generalizes both vanilla BP and DBP.

solution $\tau^* = (\tau^*_1, \ldots, \tau^*_{|X|}) \in \Pi_{x_i \in X} D_i$ such that the total cost is minimized:

$$\tau^* = \underset{\tau \in \Pi_{x_i \in X} D_i}{\arg\min} \sum_{f_\ell \in F} f_\ell(\tau|_{scp(f_\ell)}), \tag{1}$$

where $\tau|_{scp(f_\ell)}$ is the projection of τ on $scp(f_\ell)$. A COP can be represented by a factor graph which is a bipartite graph consisting of variable-nodes and function-nodes. Variable-nodes correspond to the variables, and function-nodes correspond to the constraint functions in a COP. An edge between a variable-node and a function-node is established if the variable belongs to the scope of the function.

2.2 Min-sum Belief Propagation

Min-sum Belief Propagation (Min-sum BP) [14] is an important algorithm for COPs which performs message-passing on factor graphs. More specifically, the message sent from a variable-node x_i to its neighbor f_ℓ in iteration t is computed by

$$\mu^t_{i\to\ell} = \sum_{f_m \in \mathcal{N}_i \setminus \{f_\ell\}} \mu^{t-1}_{m\to i}, \tag{2}$$

where $\mathcal{N}_i$ is the neighbors of x_i in the factor graph, and $\mu^{t-1}_{m\to i}$ is the message sent from f_m to x_i in the previous iteration. Similarly, a function-node f_ℓ computes a message for its neighbor x_i:

$$\mu^t_{\ell\to i} = \min_{\mathcal{N}_\ell \setminus \{x_i\}} \left(f_\ell + \sum_{x_j \in \mathcal{N}_\ell \setminus \{x_i\}} \mu^{t-1}_{j\to\ell} \right). \tag{3}$$

Finally, variable-node x_i makes a decision by choosing a value that minimizes the current belief:

$$\tau^t_i = \underset{\tau_i \in D_i}{\arg\min} \sum_{f_\ell \in \mathcal{N}_i} \mu^t_{\ell\to i}(\tau_i). \tag{4}$$

However, vanilla Min-sum BP usually suffers from non-convergence and explores low-quality solutions on cyclic COPs due to excessive loopy propagation. Damped BP (DBP) [8] attempts to alleviate the issues by damping the messages sent from variable-nodes to function-nodes. Formally,

$$\mu^t_{i\to\ell} = \lambda \mu^{t-1}_{i\to\ell} + (1-\lambda) \sum_{f_m \in \mathcal{N}_i \setminus \{f_\ell\}} \mu^{t-1}_{m\to i}. \tag{5}$$

By choosing a suitable damping factor $\lambda \in (0, 1]$, DBP can drastically increase the chance of convergence as well as the performance of vanilla Min-sum BP.

3 Deep Attentive Belief Propagation

Although message damping has proved to be an effective technique to improve the performance of BP on COPs [8], it still suffers from various issues. First, selecting a damping factor is largely accomplished by empirical tuning, which could be laborious and the number of trials is limited by computational resources and/or runtime. Second, using a homogeneous and static damping factor for all variable-nodes may limit DBP's capability. Finally, DBP simply treats each variable-node's neighbors equally when composing a new message, which fails to explore optimal message composition strategy.

We address the above limitations by allowing dynamic damping factors and different neighbor weights for each variable-node. More specifically, in our framework a variable-node x_i computes a message to function-node f_ℓ at iteration t by

$$\mu^t_{i\to\ell} = \lambda^t_{i\to\ell}\mu^{t-1}_{i\to\ell} + (1-\lambda^t_{i\to\ell})(|\mathcal{N}_i|-1)\sum_{f_m\in\mathcal{N}_i\setminus\{f_\ell\}} w^t_{m\to i}(\ell)\mu^{t-1}_{m\to i}, \tag{6}$$

where $\lambda^t_{i\to\ell}\in[0,1]$ and $w^t_{m\to i}(\ell)\in[0,1]$ are learnable damping factor and neighbor weights for the message from x_i to f_ℓ at iteration t, respectively. Particularly, we require $\sum_{f_m\in\mathcal{N}_i\setminus\{f_\ell\}} w^t_{m\to i}(\ell)=1$ and use $|\mathcal{N}_i|-1$ to rescale the weighted sum of the messages from neighbors. Consequently, we have more fine-grained control for composing new messages to trigger effective exploration without altering the semantic of BP messages. Fig. 1 compares three different BP variants.

A key challenge of our framework is to determine the values of a large number of *time-varying* damping factors and neighbor weights (cf. Eq. (6)). Given a factor graph with $|X|$ variables and maximum degree of d, running for T iterations would require to select $O(T|X|d^2)$ hyperparameters from the continuous range of $[0,1]$, which obviously cannot be done by the traditional trial-based hyperparameter tuning methods (e.g., grid search, evolutionary optimization).

Therefore, we propose to automatically infer the optimal hyperparameters for Eq. (6) in each iteration by seamlessly integrating BP and DNNs within the parallel message-passing framework. Our model, Deep Attentive Belief Propagation (DABP), directly outputs the optimal damping factors and weights by learning to attend neighbors given the factor graph and the BP messages in each iteration:

$$\boldsymbol{w^t}, \boldsymbol{\lambda^t} = \mathcal{M}_\theta\left(FG, \boldsymbol{\mu^{1:t-1}_{x\to f}}, \boldsymbol{\mu^{1:t-1}_{f\to x}}\right), \tag{7}$$

where FG is the factor graph, $\boldsymbol{\mu^{1:t-1}_{x\to f}}$ and $\boldsymbol{\mu^{1:t-1}_{f\to x}}$ are the BP messages from variable-nodes to function-nodes and from function-nodes to variable-nodes in the previous iterations, respectively.

However, implementing such DNN-based model requires to perform inference along with BP message-passing iterations (cf. Eq. (7)), which could incur a severe gradient vanishing or gradient exploding problem in a long reasoning sequence. Moreover, obtaining optimal training labels can be extremely expensive for COPs due to the heavy computation of exact solvers, making it impracticable to perform supervised model training on large instances. In Sect. 3.1, we address the first problem by introducing GRUs as a part of our encoder; while we design a novel self-supervised loss function and an efficient online learning framework in Sect. 3.2 to address the second issue.

3.1 Model Architecture

Fig. 2 gives the overall architecture of our DABP. It consists of an encoder to embed the factor graph and the BP messages, and an attention module to infer the dynamic optimal hyperparameters.

Encoder. The purpose of our encoder is to embed the factor graph given a sequence of BP messages (cf. Fig. 2(a-d)). Therefore, to capture the BP message dynamics and avoid gradient vanishing/exploding issue, we first use two GRUs to embed the BP messages from variable-nodes to function-nodes and the ones from function-nodes to variable-nodes into q-dimensional vectors, respectively (cf. Fig. 2(b)):

$$\boldsymbol{h^t_{x\to f}} = GRU_{\phi_1}(\boldsymbol{h^{t-1}_{x\to f}}, \boldsymbol{\mu^{t-1}_{x\to f}}), \quad \boldsymbol{h^t_{f\to x}} = GRU_{\phi_2}(\boldsymbol{h^{t-1}_{f\to x}}, \boldsymbol{\mu^{t-1}_{f\to x}}), \tag{8}$$

where $\boldsymbol{h^t_{x\to f}}, \boldsymbol{h^t_{f\to x}} \in \mathbb{R}^{r\times q}$ are the message hiddens for iteration t and r is the number of edges in the factor graph.

To consider both the topology of a factor graph and the message dynamics, we insert a set of *message nodes* that carry the message hiddens (i.e., the brown and green dots in Fig. 2(c)) and make the graph directed to reflect the message-passing directions. Then the augmented factor graph is embedded through G layers of GAT (cf. Fig. 2(d)). Formally, in the $g\in\{1,\dots,G\}$-th layer GAT_{ψ_g}, we

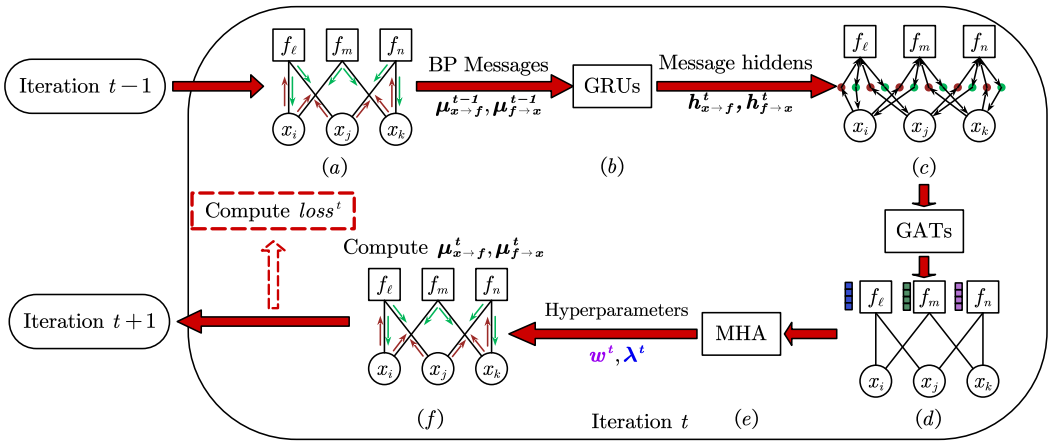


Figure 2: The overall architecture of DABP: (a) The BP messages from iteration $t-1$; (b) To update the message hidden vectors with the messages from iteration $t-1$. Note that these message hidden vectors capture the dynamics of a sequence of BP messages; (c) To insert message nodes with the hidden vectors into the factor graph; (d) To embed the augmented factor graph; (e) To infer optimal hyperparameters by using a multi-head attention layer (MHA); and (f) To compose and propagate new messages with updated hyperparameters and compute training loss in iteration t.

compute the embedding $e_i^{t,(g)}$ for node i according to

$$e_i^{t,(g)} = \text{LeakyReLU}\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}^{(g)} \boldsymbol{W}^{(g)} e_j^{t,(g-1)}\right), \quad \alpha_{ij}^{(g)} = \frac{\exp(s_{ij}^{(g)})}{\sum_{j'\in\mathcal{N}_i}\exp(s_{ij'}^{(g)})}, \tag{9}$$

where $\boldsymbol{W}^{(g)} \in \psi_g$ is a learnable matrix, $s_{ij}^{(g)} = a^{(g)}(\boldsymbol{W}^{(g)} e_i^{t,(g-1)}, \boldsymbol{W}^{(g)} e_j^{t,(g-1)})$ is the attention score between nodes i and j, and $a^{(g)}$ is a single-layer feed-forward neural network.

Attention Module. Given the embedding $e_\ell^{t,(G)}$ for each $f_\ell \in F$, we update the damping factors and neighbor weights by using a multi-head attention (MHA) layer [37] (cf. Fig. 2(e)), where queries and keys are the embeddings of the target and corresponding neighboring function-nodes, respectively. Specifically, for each variable-node x_i and a target function-node $f_\ell \in \mathcal{N}_i$, we first compute an attention score for each neighboring function-node $f_m \in \mathcal{N}_i$ in $k \in \{1,\ldots,K\}$-th head Att_{ζ_k} by:

$$a_m^{t,(k)}(\ell) = \sigma\left(\boldsymbol{W_1^{(k)}}\left[\boldsymbol{W_2^{(k)}} e_\ell^{t,(G)} || \boldsymbol{W_3^{(k)}} e_m^{t,(G)}\right]\right), \tag{10}$$

where σ is the Sigmoid function, $||$ is the concatenation operation and $\boldsymbol{W_1^{(k)}}, \boldsymbol{W_2^{(k)}}, \boldsymbol{W_3^{(k)}} \in \zeta_k$ are learnable matrices, respectively. Then the attention scores are normalized to compute neighbor weights respect to target function-node f_ℓ:

$$w_{m\to i}^{t,(k)}(\ell) = \frac{\exp\left(a_m^{t,(k)}(\ell)\right)}{\sum_{f_{m'}\in\mathcal{N}_i\backslash\{f_\ell\}}\exp\left(a_{m'}^{t,(k)}(\ell)\right)}, \quad \forall f_m \in \mathcal{N}_i\backslash\{f_\ell\}. \tag{11}$$

Similarly, the damping factor is computed by normalizing the attention score of f_ℓ w.r.t. the mean attention score of the remaining neighboring function-nodes:

$$\lambda_{i\to\ell}^{t,(k)} = \frac{\exp\left(a_\ell^{t,(k)}(\ell)\right)}{\exp\left(a_\ell^{t,(k)}(\ell)\right) + \exp\left(\frac{1}{|\mathcal{N}_i|-1}\sum_{f_{m'}\in\mathcal{N}_i\backslash\{f_\ell\}} a_{m'}^{t,(k)}(\ell)\right)} \tag{12}$$

Finally, both the neighbor weights and damping factors are normalized across different heads, i.e.,

$$\lambda_{i\to\ell}^t = \frac{\sum_{k=1}^K \lambda_{i\to\ell}^{t,(k)}}{K}, \quad w_{m\to i}^t(\ell) = \frac{\sum_{k=1}^K w_{m\to i}^{t,(k)}(\ell)}{K}, \ \forall f_m \in \mathcal{N}_i\backslash\{f_\ell\}, \tag{13}$$

and are plugged into Eq.(6) to compute new BP messages for iteration t (cf. Fig. 2(f)).

Algorithm 1 A self-supervised online learning algorithm for the DABP model $\mathcal{M}_\theta$

Input: A COP instance $\mathcal{P} = \langle X, D, F \rangle$, the number of restart R, the maximal iteration limit $T_{\max}$, the update interval $T_{\text{upd}} \leq T_{\max}$, and the number of effective iterations $T_{\text{eff}} \in \{1, \ldots, T_{\text{upd}}\}$

1: $\tau^* \leftarrow$ nil; $FG \leftarrow$ factor graph of $\mathcal{P}$
2: **for** $rs = 1, \ldots, R$ **do**
3: $\quad \mu^0_{i \to \ell} \leftarrow \mathbf{0}, \mu^0_{\ell \to i} \leftarrow \mathbf{0}, \forall x_i \in X, f_\ell \in \mathcal{N}_i$
4: $\quad$ **for** $t = 1, \ldots, T_{\max}$ **do**
5: $\quad\quad \boldsymbol{w^t}, \boldsymbol{\lambda^t} \leftarrow \mathcal{M}_\theta \left(FG, \boldsymbol{\mu^{t-1}_{x \to f}}, \boldsymbol{\mu^{t-1}_{f \to x}} \right)$
6: $\quad\quad$ compute $\mu^t_{i \to \ell}$ according to Eq. (6) with hyperparameters $\boldsymbol{w^t}, \boldsymbol{\lambda^t}, \forall x_i \in X, f_\ell \in \mathcal{N}_i$
7: $\quad\quad$ compute $\mu^t_{\ell \to i}$ according to Eq. (3) $\forall x_i \in X, f_\ell \in \mathcal{N}_i$
8: $\quad\quad$ compute solution $\tau^t = \langle \tau^t_1, \ldots, \tau^t_{|X|} \rangle$ according to Eq. (4)
9: $\quad\quad$ **if** cost(τ^*)>cost(τ^t) **then** $\tau^* \leftarrow \tau^t$
10: $\quad\quad$ **if** $t \ \% \ T_{\text{upd}} = 0$ **then**
11: $\quad\quad\quad T^* \leftarrow$ select the best T_{eff} iterations according to cost($\tau^{t'}$), $t - T_{\text{upd}} < t' \leq t$
12: $\quad\quad\quad$ compute $\mathcal{L}(\theta)$ according to Eq. (16), $\theta \leftarrow$ADAM($\theta, \nabla \mathcal{L}(\theta)$)
13: $\quad\quad$ **if** BP messages converge **then break**
return τ^*

3.2 A Novel Self-supervised Learning Algorithm for Solving COPs

Recall that our objective is to find a cost-optimal solution (cf. Eq. (1)), a nature choice of the loss function would be the cost of the solution induced by BP messages in each iteration. Unfortunately, such loss function is not differentiable since variables are greedily assigned via `argmin` (cf. Eq. (4)). Instead, we propose to use a smoothed cost as the surrogate objective. Formally, for each iteration t, we consider the following self-supervised objective:

$$loss^t = \sum_{f_\ell \in F} \left(\sum_{\langle \tau_{i_1}, \ldots, \tau_{i_n} \rangle \in \Pi_{x_i \in scp(f_\ell)} D_i} f_\ell(\tau_{i_1}, \ldots, \tau_{i_n}) \prod_{j=1:n} p^t_{i_j}(\tau_{i_j}) \right), \tag{14}$$

where $p^t_i(\tau_i)$ is the probability of $x_i = \tau_i$ under the current belief, i.e.,

$$p^t_i(\tau_i) = \frac{\exp(-b^t_i(\tau_i))}{\sum_{\tau'_i \in D_i} \exp(-b^t_i(\tau'_i))}, \quad b^t_i(\tau_i) = \sum_{f_\ell \in \mathcal{N}_i} \mu^t_{\ell \to i}(\tau_i). \tag{15}$$

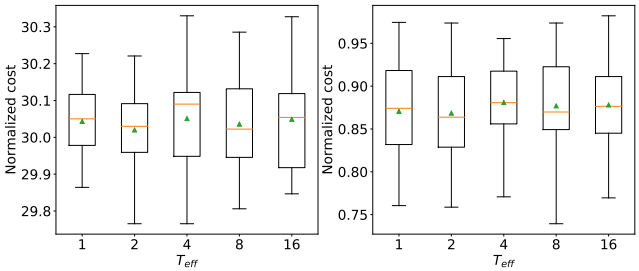
Intuitively, an assignment τ_i is associated with a high probability if it has a low belief cost $b^t_i(\tau_i)$, which simulates the behavior of `argmin` in Eq. (4) and is in alignment with the objective in Eq. (1). Given the smoothed objective for each iteration, we define the following loss function:

$$\mathcal{L}(\theta) = \frac{1}{|T^*|} \sum_{t \in T^*} loss^t, \tag{16}$$

where $\theta = \{\phi_1, \phi_2, [\psi_1, \ldots, \psi_G], [\zeta_1, \ldots, \zeta_K]\}$ is the model parameters and T^* is a selected subset of iterations used for optimizing, respectively. Particularly, we rank each iteration t according to the quality of solution τ^t and select the best T_{eff} iterations as effective iterations T^*. This way, we encourage exploration by considering only good-enough solutions.

Since our model is self-supervised by the smoothed cost, there is no need for costly label generation for each instance and thus, DABP can be directly applied to solve COPs via online learning. That is, for each COP to be solved, we continuously improve DABP by training on the instance multiple rounds (i.e., *restart*) and return the best solution as the result. Therefore, our DABP avoids the out-of-distribution issue and has a nice anytime property [42]. The procedure is detailed in Alg. 1.

Specifically, each round of restart begins with resetting BP messages to zero vectors (line 3), followed by a sequence of message-passing iterations (line 4-13). In each iteration, we query our DABP for the damping factors and neighbor weights (line 5), perform BP message passing according to the updated hyperparameters (line 6-7), and finally update the current best solution (line 8-9). Particularly, we perform model update every T_{upd} iterations (line 10-12) and thus, we further alleviate the gradient vanishing/exploding issue by restricting the maximum length of reasoning sequence to T_{upd}. Lastly, we early stop each round whenever BP messages converge (line 13) and return the best solution.



(a) Random COPs (b) WGCPs

Figure 3: Solution quality when varying T_{eff} (i.e., the number of effective iterations in Alg. 1).

4 Empirical Evaluations

In this section, we perform extensive empirical studies. We begin with introducing the details of experiments and implementation. Then we analyze the impact of the number of selected effective iterations in Alg. 1. Finally, we show the great superiority of our DABP over the state-of-the-arts.

Benchmarks & Baselines. We consider four types of standard benchmarks in our experiments, i.e., random COPs, scale-free networks, small-world networks, and Weighted Graph Coloring Problems (WGCPs). For random COPs and WGCPs, constraints are randomly established according to graph density $p_1 \in (0, 1]$. For scale-free networks, we use the BA model [3] with parameters $m_0, m_1 \in \mathbb{Z}_+$ to generate constraints. Finally, we generate constraints for small-world networks by using Newman-Watts-Strogatz model [27] with parameters $k \in \mathbb{Z}_+, p \in [0, 1]$.

For baselines, we compare our DABP with the following state-of-the-art COP solvers: (1) DBP with a damping factor of 0.9 and its splitting constraint factor graph version (DBP-SCFG) with a splitting ratio of 0.95 [8]; (2) GAT-PCM-LNS with a destroy probability of 0.2 [12]; (3) Mini-bucket Elimination (MBE) with an i-bound of 9 [11], and (4) Toulbar2 with timeout of 1200s [9].

All experiments are conducted on an Intel i9-9820X workstation with GeForce RTX 3090 GPUs and 384GB memory. *We report the best solution cost for each run, and for each experiment we average the results over 100 random problem instances*. Due to the space constraint, we defer the details of benchmarks and baselines to Appendix B.1 and B.2, respectively.

Implementation. Our DABP consists of two GRUs which embeds BP messages into 8-dimensional hidden vectors, followed by $G = 4$ layers of GAT, each of them has 8 output channels and 4 attention-heads. We then infer the optimal damping factors and neighbor weights by using a multi-head attention layer with $K = 4$ attention-heads. Finally, we also adopt the SCFG scheme with the splitting ratio of 0.95 [8]. Our model was implemented with the PyTorch Geometric framework [15] and the model was trained with the Adam optimizer [18] using a learning rate of 10^{-4} and a weight decay ratio of 5×10^{-5}. For each instance, we perform $R = 20$ restarts with iteration limit $T_{\max} = 1000$ and update the model every $T_{\text{upd}} = 20$ iterations.

Impact of the Number of Effective Iterations T_{eff}. We train our DABP with different T_{eff} on the Random COPs and WGCPs with 80 variables and $p_1 = 0.25$. Fig. 3 presents the solution quality when varying T_{eff}. It can be seen that a large T_{eff} usually produces inferior solutions. In such scenario, T^* may contain many suboptimal iterations and their solutions are relatively easy to be improved. Consequently, DABP is more exploitative and thus, it is prone to get trapped in local optima. In contrast, a small T_{eff} forces DABP to improve only good-enough solutions and encourages exploration. In our following experiments, we use $T_{\text{eff}} = 2$ due to its better solution quality.

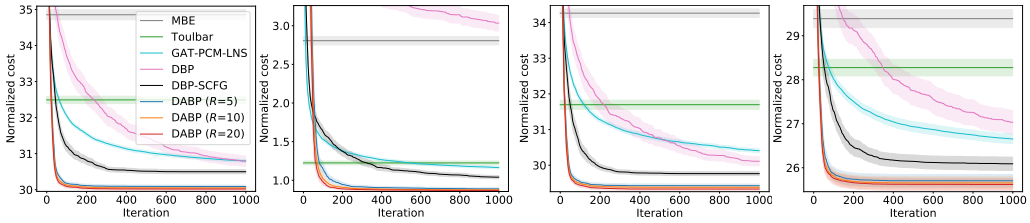
Performance Comparison. Table 1 compares the performance of different methods on the four standard benchmarks. We observe that, although given a relatively high timeout (i.e., 20 minutes),

Table 1: The performance of different methods. Costs are normalized by the number of constraints $|F|$. Gap is the ratio of cost difference w.r.t. the best result. The best results are shown in **bold**.

Methods	$\|X\| = 60$ Cost	Gap	Time	$\|X\| = 80$ Cost	Gap	Time	$\|X\| = 100$ Cost	Gap	Time
Random COPs ($p_1 = 0.25$)									
Toulbar2	29.26	7.88%	20m	32.49	8.23%	20m	34.36	7.23%	20m
MBE	32.04	18.11%	4m2s	34.85	16.10%	8m38s	36.76	14.72%	11m58s
GAT-PCM-LNS	28.00	3.21%	5m27s	30.80	2.59%	12m55s	32.78	2.31%	24m47s
DBP	27.86	2.72%	1m11s	30.79	2.58%	2m25s	33.06	3.17%	4m18s
DBP-SCFG	27.60	1.77%	**52s**	30.50	1.60%	2m6s	32.45	1.28%	3m59s
DABP ($R = 5$)	27.19	0.24%	1m4s	30.09	0.23%	**1m20s**	32.12	0.26%	**1m49s**
DABP ($R = 10$)	27.16	0.12%	2m2s	30.05	0.10%	2m41s	32.07	0.10%	3m37s
DABP ($R = 20$)	**27.12**	**0.00%**	4m	**30.02**	**0.00%**	5m20s	**32.04**	**0.00%**	7m20s
WGCPs ($p_1 = 0.25$)									
Toulbar2	**0.18**	**0.00%**	20m	1.23	41.06%	20m	2.11	42.09%	20m
MBE	1.98	1028.69%	**0s**	2.81	223.13%	**0s**	3.43	131.05%	**1s**
GAT-PCM-LNS	0.51	191.35%	57s	1.16	33.98%	2m40s	1.79	20.47%	5m56s
DBP	1.78	913.53%	29s	3.03	249.36%	1m4s	3.79	154.99%	1m55s
DBP-SCFG	0.40	130.39%	30s	1.04	19.77%	1m59s	1.69	13.97%	4m29s
DABP ($R = 5$)	0.32	80.52%	1m9s	0.89	2.39%	2m53s	1.57	5.50%	5m47s
DABP ($R = 10$)	0.30	73.80%	2m15s	0.88	0.92%	5m37s	1.50	0.69%	11m19s
DABP ($R = 20$)	0.29	67.12%	4m26s	**0.87**	**0.00%**	11m10s	**1.49**	**0.00%**	22m35s
Scale-free networks ($m_0 = m_1 = 10$)									
Toulbar2	31.01	7.51%	20m	31.70	8.07%	20m	32.69	10.51%	20m
MBE	33.70	16.85%	4m8s	34.26	16.82%	5m43s	34.58	16.91%	7m9s
GAT-PCM-LNS	29.62	2.70%	6m25s	30.41	3.66%	10m55s	31.10	5.16%	17m7s
DBP	29.32	1.65%	1m16s	30.11	2.66%	2m8s	30.47	3.04%	2m57s
DBP-SCFG	29.27	1.46%	1m18s	29.76	1.47%	1m51s	30.01	1.48%	2m27s
DABP ($R = 5$)	28.93	0.30%	**1m2s**	29.43	0.32%	**1m15s**	29.67	0.33%	**1m18s**
DABP ($R = 10$)	28.87	0.10%	2m2s	29.38	0.17%	2m33s	29.62	0.15%	2m37s
DABP ($R = 20$)	**28.84**	**0.00%**	4m3s	**29.33**	**0.00%**	5m9s	**29.58**	**0.00%**	5m12s
Small-world networks ($k = 10, p = 0.3$)									
Toulbar2	27.98	8.72%	20m	28.28	10.37%	20m	28.36	10.60%	20m
MBE	29.67	15.26%	2m46s	29.39	14.71%	3m39s	29.54	15.17%	4m50s
GAT-PCM-LNS	26.75	3.93%	4m	26.65	4.04%	6m19s	26.68	4.02%	9m12s
DBP	26.77	4.02%	**1m**	27.03	5.52%	**1m23s**	27.40	6.83%	**1m50s**
DBP-SCFG	26.30	2.17%	1m4s	26.09	1.83%	1m31s	26.13	1.90%	2m10s
DABP ($R = 5$)	25.84	0.39%	1m15s	25.70	0.33%	1m31s	25.73	0.34%	1m56s
DABP ($R = 10$)	25.78	0.15%	2m36s	25.65	0.14%	3m6s	25.70	0.20%	3m49s
DABP ($R = 20$)	**25.74**	**0.00%**	5m8s	**25.62**	**0.00%**	6m20s	**25.65**	**0.00%**	7m31s

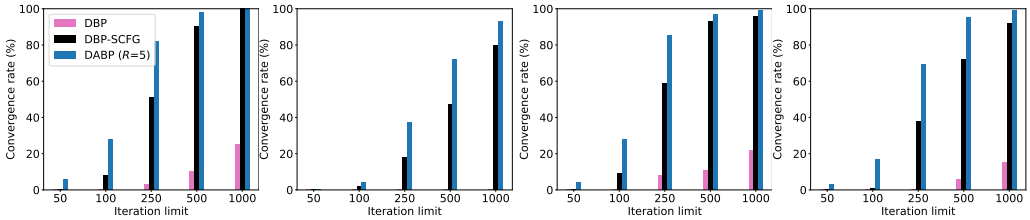
Toulbar2 still performs poorly on the most test cases. This is not surprising because it systematically explores the whole solution space, which is often infeasible for large problem instances. Also, it highlights the extreme difficulty of obtaining optimal labels with an exact solver for the existing DNN-based BP variants. MBE, on the other hand, performs memory-bounded inference and thus runs much faster than Toulbar. However, MBE fails to find good solutions and is strictly dominated by the other algorithms. GAT-PCM-LNS performs iterative large neighborhood search with a machine-learned repair heuristic and often outperforms DBP. However, GAT-PCM-LNS requires significant longer runtime than DBP since it needs to perform multiple times of model inference in each iteration to re-assign the destroyed variables. With a small modification in a factor graph, DBP-SCFG drastically improves the performance of DBP but it is computationally heavier. That is because the number of function-nodes in DBP-SCFG is doubled due to constraint function splitting.

Our DABP exhibits great superiority in various benchmarks. Specifically, given 5 times of restart, DABP substantially outperforms DBP-SCFG, which is currently the strongest approximate solver for COPs, on all test cases and the gap is widened with the increasing R. This demonstrates the merits of our learned dynamic damping factors and neighbor weights over the static and homogeneous counterpart. Importantly, our DABP's runtime performance is comparable to DBP-SCFG and also scales up well to large problem instances, thanks to our seamlessly integration of BP and DNNs within the message-passing framework which enables efficient parallel computation on GPUs. In fact, when applied to scale-free networks, DABP ($R = 5$) requires less runtime for all configurations,



(a) Random COPs (b) WGCPs (c) SF networks (d) SW networks

Figure 4: Solution quality comparison. Shaded areas show the 95% confidence intervals.



(a) Random COPs (b) WGCPs (c) SF networks (d) SW networks

Figure 5: Convergence rates under different iteration limits.

and has a much lower growing rate ($\sim$23.81%) than DBP-SCFG ($\sim$37.37%). Besides, equipped with the self-supervised loss function for minimizing the smoothed cost, our DABP is able to infer the optimal hyperparameters that foster fast convergence, which also improves our model's efficiency.

Further Performance Analysis. We further compare the solution quality of different methods in each iteration and analyze the convergence rate of BP variants on the problem instances with $|X| = 80$[2] in Fig. 4 and Fig. 5, respectively. We omit the cases of DABP with $R = 10$ and $R = 20$ in Fig. 5 since they are similar to the one of DABP ($R = 5$). It can be seen that DBP improves slowly and fails to find good solutions within 1000 iterations, especially on WGCPs. DBP-SCFG substantially improves the convergence as well as solution quality of DBP by introducing another degree of freedom through asymmetric function splitting [8]. On the other hand, our DABP allows more flexibility in composing BP messages by reasoning optimal dynamic hyperparameters through seamlessly integrating BP and DNNs, and it also significantly outperforms and converges much faster than both DBP and DBP-SCFG.

5 Conclusion

In this paper, we propose DABP, the first self-supervised DNN-based BP for solving COPs. By allowing dynamic damping factors and different neighbor weights for each variable-node, our DABP strictly generalizes the state-of-the-art DBP and has more fine-grained control for composing new messages to trigger effective exploration without altering the semantic of BP messages. Moreover, by seamlessly integrating BP, GRUs and GATs within the massage-passing framework, DABP is able to infer the optimal hyperparameters automatically according to the BP message dynamics. Finally, equipped with a novel self-supervised loss function, our DABP does not require expensive training labels and can avoid out-of-distribution issue with efficient online learning. Extensive experiments confirm the great superiority of our DABP over the state-of-the-art baselines.

Since it relies smoothed costs as the self-supervised loss function, our DABP is dedicated to solve COPs and is not trivial to be adapted for other scenarios. Therefore, in future we will explore more training paradigms to extend our model to solve other important reasoning tasks over graphical models, e.g., computing partition function [40], and finding constrained most probable explanation [34].

[2]The results for the instances with $|X| = 60$ and $|X| = 100$ can be found in Appendix B.3

References

[1] Srinivas M Aji and Robert J McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

[3] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[4] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

[5] Dimitri P Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press, 2014.

[6] Ziyu Chen, Yanchen Deng, Tengfei Wu, and Zhongshi He. A class of iterative refined max-sum algorithms via non-consecutive value propagation strategies. *Autonomous Agents and Multi-Agent Systems*, 32(6):822–860, 2018.

[7] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*, pages 1724–1734, 2014.

[8] Liel Cohen, Rotem Galiki, and Roie Zivan. Governing convergence of max-sum on DCOPs through damping and splitting. *Artificial Intelligence*, 279:103212, 2020.

[9] S. de Givry D. Allouche and T. Schiex. Toulbar2, an open source exact cost function network solver. Technical report, INRA, 2010.

[10] Rina Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *Learning in Graphical Models*, volume 89 of *NATO ASI Series*, pages 75–104. 1998.

[11] Rina Dechter and Irina Rish. Mini-buckets: A general scheme for bounded inference. *Journal of the ACM*, 50(2):107–153, 2003.

[12] Yanchen Deng, Shufeng Kong, and Bo An. Pretrained cost model for distributed constraint optimization problems. *arXiv preprint arXiv:2112.04187*, 2021.

[13] Yanchen Deng, Runsheng Yu, Xinrun Wang, and Bo An. Neural regret-matching for distributed constraint optimization problems. In *IJCAI*, pages 146–153, 2021.

[14] Alessandro Farinelli, Alex Rogers, Adrian Petcu, and Nicholas R Jennings. Decentralised coordination of low-power embedded devices using the Max-sum algorithm. In *AAMAS*, pages 639–646, 2008.

[15] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[16] Khoi D Hoang, Ferdinando Fioretto, William Yeoh, Enrico Pontelli, and Roie Zivan. A large neighboring search schema for multi-agent optimization. In *CP*, pages 688–706, 2018.

[17] Yoonheui Kim, Michael Krainin, and Victor Lesser. Effective variants of the max-sum algorithm for radar coordination and scheduling. In *WI/IAT*, pages 357–364, 2011.

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

[19] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *ICLR*, 2019.

[20] Frank R Kschischang, Brendan J Frey, and H-A Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.

[21] Jonathan Kuck, Shuvam Chakraborty, Hao Tang, Rachel Luo, Jiaming Song, Ashish Sabharwal, and Stefano Ermon. Belief propagation neural networks. In *NeurIPS*, pages 667–678, 2020.

[22] Eugene L Lawler and David E Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, 1966.

[23] Kathryn S. Macarthur, Ruben Stranders, Sarvapali D. Ramchurn, and Nicholas R. Jennings. A distributed anytime algorithm for dynamic task allocation in multi-agent systems. In *AAAI*, pages 701–706, 2011.

[24] David JC MacKay and David JC Mac Kay. *Information theory, Inference and Learning Algorithms*. Cambridge University Press, 2003.

[25] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1-2):149–180, 2005.

[26] Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

[27] Mark EJ Newman and Duncan J Watts. Renormalization group analysis of the small-world network model. *Physics Letters A*, 263(4-6):341–346, 1999.

[28] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan kaufmann, 1988.

[29] Yasaman Razeghi, Kalev Kask, Yadong Lu, Pierre Baldi, Sakshi Agarwal, and Rina Dechter. Deep bucket elimination. In *IJCAI*, pages 4235–4242, 2021.

[30] Alex Rogers, Alessandro Farinelli, Ruben Stranders, and Nicholas R Jennings. Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence*, 175(2):730–759, 2011.

[31] Emma Rollon and Javier Larrosa. Improved bounded max-sum for distributed constraint optimization. In *CP*, pages 624–632, 2012.

[32] Emma Rollon and Javier Larrosa. Decomposing utility functions in bounded max-sum for distributed constraint optimization. In *CP*, pages 646–654, 2014.

[33] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming*. Elsevier Science Inc., USA, 2006.

[34] Sara Rouhani, Tahrima Rahman, and Vibhav Gogate. A novel approach for constrained optimization in graphical models. In *NeurIPS*, pages 11949–11960, 2020.

[35] Victor Garcia Satorras and Max Welling. Neural enhanced belief propagation on factor graphs. In *AISTATS*, pages 685–693, 2021.

[36] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *CP*, pages 417–431, 1998.

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017.

[38] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[39] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *NeurIPS*, pages 2692–2700, 2015.

[40] Jonathan S Yedidia, William T Freeman, and Yair Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312, 2005.

[41] Zhen Zhang, Fan Wu, and Wee Sun Lee. Factor graph neural networks. In *NeurIPS*, pages 8577–8587, 2020.

[42] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–73, 1996.

[43] Roie Zivan, Omer Lev, and Rotem Galiki. Beyond trees: Analysis and convergence of belief propagation in graphs with multiple cycles. In *AAAI*, pages 7333–7340, 2020.

[44] Roie Zivan, Tomer Parash, Liel Cohen, Hilla Peled, and Steven Okamoto. Balancing exploration and exploitation in incomplete min/max-sum inference for distributed constraint optimization. *Autonomous Agents and Multi-Agent Systems*, 31(5):1165–1207, 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Sect. 3 and Sect. 4.
   (b) Did you describe the limitations of your work? [Yes] We have described our limitation in Sect. 5.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] Our work is theoretical and we do not see any potential negative societal impacts.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See the supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See the **Implementation** paragraph in page 7.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We display the box-plot or the 95% confidence intervals when presenting cost performance.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See the **Benchmarks & Baselines** paragraph in page 7 and Table 1.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Related Work

Although COP is NP-hard in general [33], many algorithms have been proposed to solve it in the last several decades, such as BP, local search [16], branch and bound [22], and bucket elimination [10]. Among them, Min-sum BP is an effective approximate algorithm for solving COPs and has been successfully applied to many real-world scenarios [14, 17, 23]. However, vanilla BP offers no convergence guarantee and usually returns low-quality solutions on problems with cyclic factor graphs. Rogers et al. proposed to ensure convergence and have a lower bound on solution quality by relaxing the problem, i.e., removing edges to have acyclic factor graphs [30] through minimization marginalization. Further, Rollon et al. improved the bounding scheme by considering both minimization and maximization marginalization [31] and decomposing high-arity factors into unary functions [32]. On the other hand, Zivan et al. proposed to avoid loopy message propagation by strictly controlling the message-passing direction [44]. They also introduced the Value Propagation (VP) scheme to enforce exploitation. To balance exploration and exploitation, a set of non-consecutive VP strategies are proposed [6] as well. Recently, Damped BP (DBP) [8] has been proposed to enhance convergence as well as solution quality in loopy BP by mixing the new messages composed in each iteration with the old messages in previous iterations with a suitable damping factor. However, selecting a good damping factor often requires expertise and laborious case-by-case tuning [43], and DBP uses a static damping factor and treats each node's neighbors equally when composing a new message, which limits its exploration. Our DABP overcomes the aforementioned issues by learning to attend neighbors with dynamic damping factors and neighbor weights when composing new BP messages. Obviously, DABP generalizes DBP since DBP can be considered as a static version of DABP. Moreover, equipped with a self-supervised learning algorithm, our DABP is able to infer the optimal damping factors and weights to enhance the performance of DBP.

Recently, there has been increasing interest in applying DNNs to boost the performance of BP. The most relevant works including: NEBP [35] learns to refine BP messages with some black-box DNN components in each iteration and is applied to solve the error correction decoding task; In the same vein, BPNN [21] uses damping layers to modify factor-to-variable messages and is applied to estimate the partition function of probabilistic graphical models (PGMs); FGNN [41] generalizes Graph Neural Networks (GNNs) to represent Max-Product BP and is applied to find approximate maximum a posteriori (MAP) assignment of a PGM. However, all these methods require supervised learning and cannot be directly applied to solve COPs when training labels are not available; They also destroy the semantic of a BP message being the weighted sum of cost functions' marginalizations. Contrarily, we only learn optimal dynamic factors and neighbor weights with DNNs to generalize DBP, which preserves the semantic of the BP messages and also enables us using a self-supervised manner to train our model. Therefore, our model does not require expensive training labels which are often not available in practice due to the heavy computation of exact solvers; and also avoids out-of-distribution issue and enjoys a nice anytime property [42] through efficient online learning.

Our method is also closely related to using DNNs to solve combinatorial problems [4, 12, 19, 26, 39]. In the context of COP solving, Neural RM-RRS [13] uses Multi-layer Perceptrons (MLPs) to parameterize high-dimensional regret tables produced by context-based regret-matching. Similarly, Deep Bucket Elimination (DBE) [29] uses MLPs to approximate the large bucket functions [10]. Recently, Deng et al. [12] proposed a pretrained cost model which predicts the optimal cost of a given partially instantiated COP. The predicted cost is then used to construct heuristics for various COP algorithms such as Large Neighborhood Search (LNS) [16] and backtracking search. These methods either replace some heavy computation components of COP algorithms by a fast DNN approximation, or learn effective heuristics (or hyperparameters) to boost the performance of COP algorithms, and our method falls into the second category.

B Additional Experimental Results

In this section, we present additional experimental details and results. Sect. B.3.

B.1 Benchmarks

We consider four types of benchmarks in our experiments, i.e., random COPs, scale-free networks, small-world networks, and Weighted Graph Coloring Problems (WGCPs). For random COPs and WGCPs, given $|X|$ variables and density of $p_1 \in (0, 1)$, we randomly create a constraint for a pair of variables with probability p_1. For scale-free networks, we use the BA model [3] with parameter m_0 and m_1 to generate constraints: starting from a connected graph with m_0 vertices, a new vertex is connected to m_1 vertices with a probability which is proportional to the degree of each existing vertex in each iteration. For small-world networks, we generate problem topology according to Newman-Watts-Strogatz model [27]: starting from a ring of $|X|$ vertices, each vertex is connected to its k nearest neighbors, and for each edge underlying the ring with k nearest neighbors, we create a new shortcut by randomly selecting another vertex with a probability p. Finally, for each variable and constraint in the benchmarks except WGCPs, we set domain size to 15 and uniformly sample a cost from $[0, 100]$ for each possible assignment combination, respectively. Differently, for WGCPs each variable has a domain of 5 values, and for any pair of constrained variables, we uniformly sample a cost from $[1, 100]$ for an unanimous variable assignment and otherwise, a zero cost.

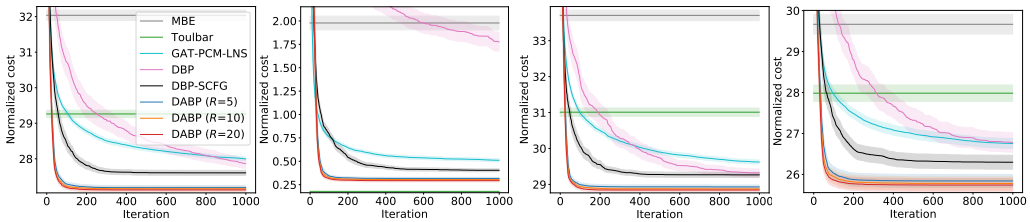
B.2 Baselines

Since BPNN [35] and NEBP [21] are proposed for partition function estimation and error correction decoding, respectively, it is not trivial to adopt them for solving COPs. Although FGNN [41] can be applied to solve COPs, all of these existing DNN-based BP variants require expensive supervised learning and optimal labels which are not available in our experimental settings because we consider large-scale problems and it is not practical to obtain optimal labels using heavy exact solvers. *Therefore, to be fair and practical, we do not consider existing DNN-based BP variants.*

We compare our DABP with the following state-of-the-art COP solvers: (1) DBP with a damping factor of 0.9 and its splitting constraint factor graph version (DBP-SCFG) with a splitting ratio of 0.95 [8]; (2) GAT-PCM-LNS with a destroy probability of 0.2 [12], which is a local search method combining the LNS framework [16, 36] with neural-learned repair heuristics; (3) Mini-bucket Elimination (MBE) with an i-bound of 9 [11], which is a memory-bounded inference algorithm; (4) Toulbar2 with timeout of 1200s [9], which is a highly optimized exact solver written in C++. The hyperparameters for DBP and GAT-PCM-LNS are set according to the original papers, while the memory budget for MBE and timeout for Toulbar2 are set based on our computational resources.

All experiments are conducted on an Intel i9-9820X workstation with GeForce RTX 3090 GPUs and 384GB memory. We terminate DBP(-SCFG) and GAT-PCM-LNS whenever convergence or the maximum iteration limit ($T_{\max} = 1000$) reaches. Finally, we report the best solution cost for each run, and for each experiment we average the results over 100 random problem instances.

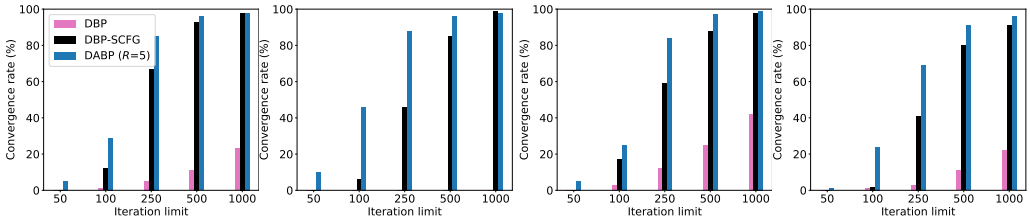
B.3 Results on the Instances with 60 and 100 Variables



(a) Random COPs (b) WGCPs (c) SF networks (d) SW networks

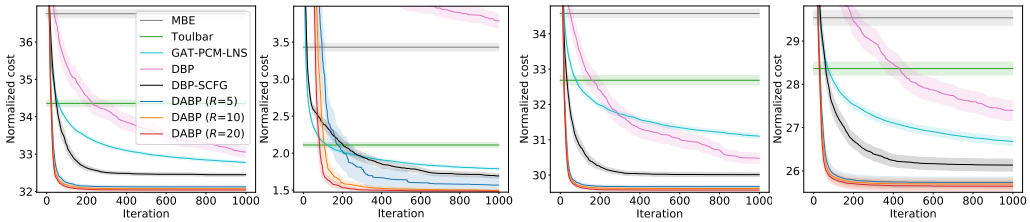
Figure 6: Solution quality comparison ($|X| = 60$)

Fig. 6-9 present solution quality in each iteration and convergence rate on the instances with $|X| = 60$ and $|X| = 100$. Our DABP outperforms the other baselines by considerable margins on a wide range of benchmarks. The only exception happens on WGCPs with 60 variables, where Toulbar2



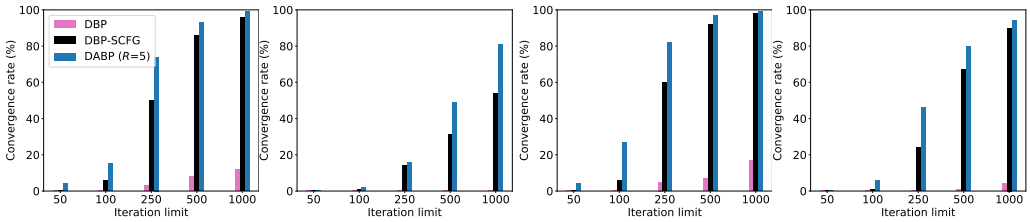
(a) Random COPs (b) WGCPs (c) SF networks (d) SW networks

Figure 7: Convergence rates under different iteration limits ($|X| = 60$)



(a) Random COPs (b) WGCPs (c) SF networks (d) SW networks

Figure 8: Solution quality comparison ($|X| = 100$)



(a) Random COPs (b) WGCPs (c) SF networks (d) SW networks

Figure 9: Convergence rates under different iteration limits ($|X| = 100$)

exhibits the best performance. That is because the variables in WGCPs have a relatively small domain size (i.e., 5) and the constraint functions are highly structured, which allows effective pruning and enumeration for exact methods like Toulbar2. It also can be concluded that our DABP converges much faster than DBP and DBP-SCFG. Notably, when solving WGCPs with 100 variables (cf. Fig. 9(b)), DBP-SCFG has a poor convergence rate and, even worse, DBP entirely fails to converge, while our DABP still achieve convergence on the most of instances, which demonstrates the virtues of our learned dynamic hyperparameters.