Noname manuscript No.
(will be inserted by the editor)

Learning Heuristics for Weighted CSPs through Deep Reinforcement Learning

Dingding Chen[1] · **Ziyu Chen**[1] · **Zhongshi He**[1] · **Junsong Gao**[1] · **Zhizhuo Su**[2]

Received: date / Accepted: date

Abstract Weighted constraint satisfaction problems (WCSPs) are one of the most important constraint programming models aiming to find a cost-minimal solution. Due to its NP-hardness, solving a WCSP usually requires efficient heuristics to explore high-quality solutions. Unfortunately, such heuristics are hand-crafted and may not be generalizable across different cases. On the other hand, although Deep Learning (DL) has been proven to be a promising way to learn heuristics for combinatorial optimization problems, the existing DL-based methods are unsuitable for WCSPs since they fail to exploit the problem structure of WCSPs. Besides, such methods are often based on Supervised Learning (SL), making the learned heuristics less efficient since it is challenging to generate a sufficiently large training corpus. Therefore, we propose a novel Deep Reinforcement Learning (DRL) framework to train the model on large-scale problems, so that the model could mine more sophisticated patterns and provide high-quality heuristics. By exploiting the problem structure, we effectively decompose the problem by using a pseudo tree, and formulate the solution construction process as a Markov decision process with multiple independent transition states. With Graph Attention Networks (GATs) parameterized deep Q-value network, we learn the optimal Q-values through a modified Bellman equation that considers the multiple transition states, and extract the solution construction heuristics from the trained network. Besides constructing solutions greedily, our heuristics can also be applied to many meta-heuristics such as beam search and large neighborhood search. The experiments show that our DRL-boosted algorithms significantly outperform the counterparts with their heuristics derived from the SL model, their counterparts with traditional tabular-based heuristics and state-of-the-art methods on benchmark problems.

🖂 Ziyu Chen
E-mail: chenziyu@cqu.edu.cn
[1]College of Computer Science, Chongqing University, Chongqing 400044, China
[2]WMG, University of Warwick, Coventry, CV47AL, United Kingdom

Keywords WCSP · Incomplete WCSP Algorithm · Heuristics · DRL · GATs

1 Introduction

Constraint satisfaction problems (CSPs) [14] provide a unified framework for general problem modeling and solving in artificial intelligence. A CSP consists of a fixed set of variables, each with an associated domain of values, and a set of hard constraints on these variables to specify the allowable assignment combinations. A CSP with a Boolean domain for each variable is also known as a Boolean Satisfiability Problem (SAT). Weighted constraint satisfaction problems (WCSPs) [47] are a generalization of CSPs where the constraints are no longer "hard". Instead, each tuple in a constraint, i.e., all variable assignments involved in that constraint, is associated with a positive real value as a weight (usually called a "cost") to specify the violation degree of constraints. The objective of solving a WCSP is to find an assignment for all variables that minimizes the aggregated constraint cost. WCSPs have been successfully applied into many real-world applications, including supply-chain management [23], judge assignment [24], bioinformatics [53,58] and many others.

Algorithms for WCSPs can be classified into two categories, i.e., complete algorithms and incomplete algorithms. Complete algorithms guarantee to find the optimal solution and can be broadly classified into inference-based algorithms [13] and search-based algorithms [33]. Search-based algorithms systematically explore the entire solution space by branch-and-bound. Instead, inference-based algorithms employ a dynamic programming paradigm to solve a WCSP. Since solving WCSPs is NP-Hard, complete algorithms exhibit an exponentially increasing computation overhead and cannot scale up to large real-world applications. Therefore, there has been considerable research effort to develop incomplete algorithms to find high-quality solutions at an acceptable computational overhead. Incomplete algorithms generally follow four strategies, i.e., suboptimal variants of the complete algorithms [15], local search [63,39,26], belief propagation [19,11] and sampling [40,38]. However, existing incomplete algorithms usually rely on hand-crafted heuristics that require domain-specific knowledge to tune for different settings, such as the large neighborhood selection policy in large neighborhood search [42] and the temperature schedule in simulated annealing [9].

Deep learning (DL) has been proven to be promising in learning algorithms for solving NP-hard problems [4,43]. Several works have tried to leverage DL to learn effective heuristics automatically for existing methods, such as the value selection heuristics for a constraint programming solver [6] and the variable selection heuristic for a stochastic local search algorithm [61]. Unfortunately, the learned heuristics are only applicable to a particular algorithm and usually cannot be generalized to other algorithms. Recently, Deng et al. [16] proposed a supervised pre-trained model to generate heuristics for a wide range of algorithms. However, generating a sufficiently large training corpus is a challenging task, which makes the model only trained on small-scale problems and thus affects the quality of the learned heuristics.

Instead, we propose to train the model on large-scale problems with Deep Reinforcement Learning (DRL) to generate efficient solution construction heuristics for WCSPs. Specifically, our main contributions are listed as follows:

– We propose a novel DRL framework where the model could be trained on large-scale problems. As a result, our model could capture more sophisticated patterns of problems to provide high-quality solution construction heuristics for WCSPs. To improve computational efficiency, we propose to use a pseudo tree to decompose the problem effectively, and formulate the solution construction process as a Markov Decision Process (MDP) with multiple independent transition states by exploiting the problem structure. In the MDP formulation, the state is the remaining sub-problem to be solved, action space is the domain of the target variable, and reward is the change of the current cost.
– We propose to use Graph Attention Networks (GATs) [56] parameterized deep Q-value network to learn the optimal Q-values through a modified Bellman Equation that considers the multiple transition states. And, the solution construction heuristics are derived from the learned Q-value network. Besides constructing greedy solutions, we embed our heuristics into meta-heuristics including Beam Search (BS) [50] and Large Neighborhood Search (LNS) [51]. Specifically, our heuristics can be used as a quality evaluation of the partial assignments in BS or as a sub-routine of LNS to repair destroyed variables via constructing a solution greedily.
– Extensive empirical evaluations indicate that the heuristics derived from our DRL model are superior to those obtained from the SL model [16]. Furthermore, we empirically demonstrate the effectiveness of our heuristics by combining them with meta-heuristics including BS and LNS on various standard benchmarks.

The rest of this paper is organized as follows. We briefly review related work in Sect. 2. The preliminaries, including WCSPs, pseudo tree, GATs, Markov Decision Process (MDP), beam search and large neighborhood search, are presented in Sect. 3. In Sect. 4, we describe our proposed method, including the MDP formulation of WCSPs, graph representation and graph embedding, the DRL training algorithm, and heuristics for beam search and large neighborhood search. Finally, we present the empirical evaluation of our proposed method in Sect. 5 and our conclusion in Sect. 6.

2 Related Work

In this section, we first review traditional tabular-based algorithms for WCSPs, and then introduce state-of-the-art deep learning-based methods for constraint reasoning.

2.1 Tabular-based Algorithms for WCSPs

The complete approaches for WCSPs employ either inference or systematic search. Branch-and-Bound (BnB) is the most significant landmark of search-based algorithm. Given a variable ordering, BnB sequentially explores the whole solution space in a depth-first fashion. Pruning happens when the current lower bound is higher than the known upper bound. Therefore, the bound quality is the key of efficient pruning. Currently, several works attempt to tighten the bounds via local consistency [32,12] or combining the benefit of the best-first search and depth-first search strategies [1]. On the other hand, bucket elimination [13,41] is a well-known algorithm for WCSPs that performs dynamic programming on the variable ordering (e.g., a pseudo tree). The algorithm forwards the assignment combination utilities bottom-up and then reversely propagates the optimal decisions along with the variable ordering. However, its memory consumption is exponential in the induced width [14]. Therefore, the memory-bound bucket elimination method [8] was proposed to trade the runtime for smaller memory consumption. Since solving WCSPs is NP-hard, complete algorithms require exponential computation overheads, making them unsuitable for large-scale practical applications.

Incomplete algorithms can be generally classified into suboptimal variants of the complete approaches, local search, belief propagation-based and sampling-based algorithms. Mini-bucket elimination [15] was proposed as an approximation version of bucket elimination to trade solution quality for smaller memory consumption. Inspired by Monte-Carlo tree search, a sampling-based algorithm with confidence bounds [40] was proposed for solving WCSPs. However, the algorithm requires an exponential memory with the number of variables, limiting its application to large-scale problems. Therefore, Nguyen et al. [38] proposed to map a WCSP to a maximum a-posteriori estimation problem and solve it by Gibbs sampling. Local search algorithms [63,39] are the most popular incomplete methods for WCSPs, where each variable optimizes its assignment based on its local constraints and assignments of all its neighbors. Unfortunately, these algorithms tend to converge to local optima. Therefore, K-optimality (K-OPT) [57] was proposed to improve the solution of local convergence by optimizing the assignments of all variables within the K-size coalition. However, the computational effort required to find the K-optimal solution grows exponentially as K increases. Recently, T-LNS [26] was proposed to solve a WCSP through combining bucket elimination and Large Neighborhood Search (LNS) [51], which is a meta-heuristic algorithm that iteratively explores a large neighborhood of the current solution to find a better one. Max-Sum [19,7] is a classic belief propagation-based algorithm that gathers global information by propagating and accumulating the maximum utility through the whole factor graph. Unfortunately, Max-Sum is only guaranteed to converge in cycle-free problems. Thus, Damped Max-Sum (DMS) [11] was proposed to improve the convergence probability of belief propagation by reducing the influence of information circulation propagation.

However, incomplete algorithms still either require considerable computational efforts (e.g., K-OPT) or exhibit poor performance (e.g., local search) in general problems. Therefore, by leveraging its powerful representational capability, we resort to deep neural networks to embed WCSPs to learn efficient and effective heuristics.

2.2 Deep Learning-based Methods for Constraint Reasoning

In recent years, methods that use deep learning techniques for constraint reasoning (i.e., solving SAT, CSPs and WCSPs) have received increasing attention, and can be generally divided into Supervised Learning (SL) based and Deep Reinforcement Learning (DRL) based methods. The classical SL-based method for SAT is NeuroSAT [49], which uses a message-passing neural network to solve the problems in an end-to-end fashion and further decode the satisfactory assignments. Later, Selsam et al. proposed NeuroCore [48], which modified NeuroSAT to guide Conflict-Driven Clause Learning (CDCL) solvers with the unsatisfiable core predictions computed by the neural network. Recently, NLocalSAT [62] was proposed to boost the performance of the stochastic local search solver by guiding initialization assignments with a gated graph convolutional network. In these methods, a SAT instance is encoded as an unweighted bipartite graph to provide permutation and variable relabeling invariance. However, unweighted bipartite graphs cannot take into account any positive real-valued constraint costs of WCSPs, which makes them unsuitable for representing WCSPs (cf. Section 4.2).

CSP-cNN [60], a SL-based method for CSPs, was proposed to encode a binary CSP as a matrix and train a Convolutional Neural Network (CNN) for predicting the satisfiability of the problems. Nonetheless, the representation scheme makes it infeasible when applying to problems with different numbers of variables. Instead, Galassi et al. [22] proposed to construct a feasible solution of a CSP instance by extending a partial assignment with a trained deep neural network. Still, this approach is restricted to problems of a pre-determined size. Differently, our state representation scheme exploits a tripartite graph to capture the complex relations among variables and constraints, and thus is able to effectively process instances of arbitrary sizes.

Deep Bucket Elimination (DBE) [44] and Neural Regret-Matching with Regret Rounding Scheme (Neural RM-RRS) [17] are early SL-based methods for solving WCSPs, which uses Multi-layer Perceptrons (MLPs) to parameterize the high-dimensional data in bucket elimination [13] and regret matching [25], respectively. Unfortunately, the online learning nature prevents these methods from generalizing to unseen instances. Furthermore, training MLPs requires considerable time. Therefore, Deng et al. [16] proposed using SL to learn a pretrained model to construct effective heuristics for a broad range of algorithms. However, generating optimally labeled data for supervised training could be infeasible in large problem instances. Different from these existing methods for WCSPs, we focus on training the model on large-scale problems with DRL, so that our model could capture more sophisticated patterns of problems to provide high-quality solution construction heuristics for solving WCSPs.

Conversely, there is another line of works leveraging DRL to enhance the existing algorithms by learning heuristics. For instance, Yolcu and Poczos [61] proposed to use Graph Neural Networks (GNNs) [46] to encode SAT instances and REINFORCE [59] to learn satisfying assignments inside a stochastic local search procedure. Song et al. [52] proposed to employ DRL to learn the variable ordering heuristics for the backtracking search algorithms, where the underlying structure of a CSP instance is represented using GNNs. Besides, learning branching heuristics for constraint programming via DRL has been intensively investigated [28,5,6]. Though all of them aim to learn the heuristics automatically with DRL, our approach differs from [61,52,5,6] in two aspects. First, our proposed DRL framework aims to learn solution construction heuristics that can be embedded in many different algorithms, while these existing methods are solely devoted to learn heuristics for a particular algorithm. Second, our models can explicitly consider the problem structure of WCSPs (e.g., the constraint costs) to make the learned heuristics more effective. However, the state representation in these existing DRL-based methods either cannot be used for WCSPs or cannot take into account the problem structure of WCSPs, as will be discussed in Section 4.2. A wider view on deep learning for constraint reasoning can be found in the more recent survey [43].

3 Background

3.1 Weighted Constraint Satisfaction Problems

A weighted constraint satisfaction problem (WCSP) can be defined by a tuple (X, D, F) such that:

- $X = \{x_1, x_2, ..., x_n\}$ is a set of variables.
- $D = \{D_1, D_2, ..., D_n\}$ is a set of finite variable domains. Each domain $D_i \in D$ consists of a set of finite allowable values for variable $x_i \in X$. Besides, we denote the maximal domain size as $d = \max_{D_i \in D} |D_i|$.
- $F = \{f_1, f_2, ..., f_m\}$ is a set of constraint functions. Each function $f_i : D_{i_1} \times D_{i_2} \times \cdots \times D_{i_k} \to \mathbb{R}_{\geq 0}$ specifies a non-negative cost to each value combination of the involved variables $x_{i_1}, x_{i_2}, ..., x_{i_k}$.

For the sake of simplicity, we assume that all constraint functions are binary (i.e., $f_{ij} : D_i \times D_j \to \mathbb{R}_{\geq 0}$). A partial assignment Γ is a set of assignments in which each variable appears at most once. The cost of Γ is calculated by aggregating the cost of all constraints involving only the variables that appear in the partial assignment. That is,

$$cost(\Gamma) = \sum_{f_{ij} \in F, x_i, x_j \in \Gamma} f_{ij}\left(\Gamma_{[x_i]}, \Gamma_{[x_j]}\right) \tag{1}$$

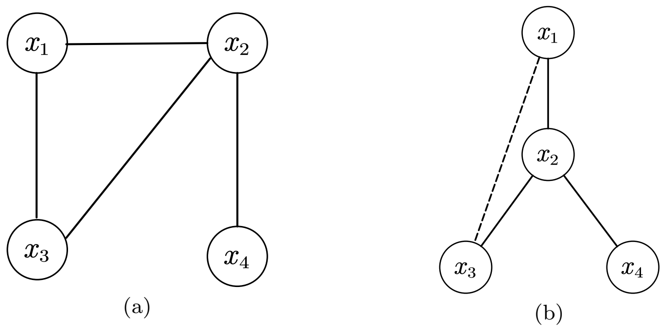


Fig. 1: An example of constraint graph and pseudo tree

where $\Gamma_{[x_i]}$ is the assignment of x_i in Γ. A partial assignment that includes all variables in X is a full assignment. The goal of solving a WCSP is to find a full assignment to minimize the aggregated constraint cost. That is,

$$X^* = \underset{d_i \in D_i, d_j \in D_j}{\arg\min} \sum_{f_{ij} \in F} f_{ij}\left(d_i, d_j\right) \tag{2}$$

A WCSP can be visualized by a constraint graph where a vertex represents a variable and an edge represents a binary constraint. Fig. 1(a) gives the constraint graph of a WCSP with four variables and four binary constraints.

3.2 Pseudo Tree

A pseudo tree [21] arrangement of a constraint graph is a tree with the same nodes and edges as the original graph. It has the property that adjacent nodes of the original graph fall in the same branch of the tree. Therefore, the sub-problems in different branches can be solved independently. A pseudo tree can be generated by a depth-first traversal of the constraint graph, categorizing the constraints into tree edges and pseudo edges (i.e., non-tree edges). Fig. 1(b) presents a possible pseudo tree derived from Fig. 1(a), where tree edges and pseudo edges are shown as solid lines and dashed lines, respectively. For a variable x_i in the pseudo tree, we denote the ancestor connected with x_i through a tree edge as its parent $P(x_i)$. The ancestors connected with x_i through back edges as its pseudo parents $PP(x_i)$, and the descendants connected with x_i through tree edges as its children $C(x_i)$. In addition, we denote all parents of x_i as $AP(x_i)$, i.e., $AP(x_i) = PP(x_i) \cup \{P(x_i)\}$. Taking x_2 in Fig. 1(b) as an example, we have $P(x_2) = x_1$, $PP(x_2) = \emptyset$, $C(x_2) = \{x_3, x_4\}$, and $AP(x_2) = \{x_1\}$.

3.3 Graph Attention Networks

Graph Attention Networks (GATs) [56] are a convolutional neural network architecture operated on graph-structured data. They are constructed by stacking several graph attention layers such that nodes can attend over the features of their neighbors. Furthermore, to indicate the importance of different nodes in the neighborhood, GATs introduce a self-attention mechanism that assigns different weights to these different nodes. In the self-attention mechanism, the calculation of the attention coefficient between each pair of neighbor nodes is:

$$e_{ij} = \mathbf{a}(\mathbf{W}h_i, \mathbf{W}h_j) \tag{3}$$

where $h_i, h_j \in \mathbb{R}^d$ are the features of nodes i, j, $\mathbf{W} \in \mathbb{R}^d \times \mathbb{R}^d$ is a weight matrix, and $\mathbf{a}$ is a single-layer feed-forward neural network.

Then, the attention coefficients are normalized using the soft-max function to facilitate comparison between different nodes. The normalization coefficient across node j is computed by:

$$\alpha_{ij} = \frac{e_{ij}}{\sum_{k \in N_i} \exp(e_{ik})} \tag{4}$$

where N_i is the neighborhood of node i in the graph (including i). These normalized attention coefficients are then computed by the linear combination of their corresponding features and served as the final output feature for each node. That is,

$$h_i' = \sigma\left(\sum_{j \in N_i} \exp(\alpha_{ij}\mathbf{W}h_j)\right) \tag{5}$$

where σ is a nonlinear function, such as LeakyReLU or sigmoid.

In order to stabilize the learning process of self-attention, GATs adopt Multi-head attention [55] in which K independent attention mechanisms are performed. Their features are averaged as:

$$h_i' = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j \in N_i} \exp\left(\alpha_{ij}^k \mathbf{W}^k h_j\right)\right) \tag{6}$$

where α_{ij}^k are normalized attention coefficients computed by the k-th attention mechanism ($\mathbf{a}_k$), and $\mathbf{W}^k$ is the corresponding input linear transformation's weight matrix.

3.4 Markov Decision Process

A Markov Decision Process (MDP) is formulated as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma \rangle$, where:

- $\mathcal{S}$ is a set of environment and agent states.

- $\mathcal{A}$ is a set of actions of the agent.
- $\mathcal{T}(s,a,s') = \Pr(s,a,s')$ is the transition function, specifying the probability of the transition from an initial state s to a new state s' under an action a. Here, $s, s' \in S$ and $a \in A$.
- $r(s,a)$ is the reward function, defining the immediate reward after taking an action a at the state s.
- $\gamma \in [0,1)$ is the discount factor, used to weight the preferences for immediate reward relative to future reward.

The agent interacts with the environment following a policy (i.e., $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0,1]$) which describes the probability of taking an action in a given state. At each decision point t, the agent observes the current state $s_t \in \mathcal{S}$, selects an action $a_t \in \mathcal{A}$ according to the policy π, and receives an immediate reward $r(s_t,a_t)$. The goal of the agent is to learn the optimal Q-function ($Q_{\pi^*}(s,a)$), an action-value function that estimates the sum of future rewards after taking an action a at the state s and following the optimal policy π^*, i.e.,

$$\begin{aligned} Q_{\pi^*}(s,a) &= \max_{\pi} Q_{\pi}(s,a) \\ &= \max_{\pi} \mathbb{E}_{\pi,\mathcal{T}} \left[r(s_t,a_t) + \gamma r(s_{t+1},a_{t+1}) + \cdots \mid s_t = s, a_t = a, \pi \right] \end{aligned} \tag{7}$$

The optimal Q-function is the fixed-point of Bellman Equation [3], i.e.,

$$Q_{\pi^*}(s,a) = \mathbb{E}_{\pi,\mathcal{T}} \left[r(s,a) + \gamma \max_{a'} Q_{\pi^*}(s',a') \right] \tag{8}$$

According to the Bellman Equation, the Q-learning algorithm [54] can be derived. However, the Q-learning algorithm suffers from the curse of dimensionality, becoming less efficient as the dimensionality of the environment increases. Therefore, Deep Q-network (DQN) [36] was proposed to use a multi-layered neural network (i.e., the main Q-network $Q_\theta(s,a)$) to parameterize the optimal Q-function. DQN can handle complicated decision processes with large and continuous state space by directly outputting the predicted Q-value of each state-action pair with state features as input.

To train the main Q-network efficiently, DQN introduces an experience replay memory and a separate target network. A First-In-First-Out (FIFO) experience replay memory is established to store the agent's experience to remove the correlation between consecutive transitions. As a result, the high variance in parameter updates and the instability of the training processes can be reduced. The update of the parameter of the main Q-network is based on a mini-batch of samples randomly drawn from the experience replay memory. The target Q-network $Q_{\bar{\theta}}(s,a)$, a copy of the main Q-network with delayed updates, is introduced to compute the target-Q values so as to stabilize the learning process [36]. The loss function of the DQN algorithm can be defined by:

$$L(\theta) = \left(Q_\theta(s,a) - r(s,a) - \gamma \max_{a'} Q_{\bar{\theta}}(s',a') \right)^2 \tag{9}$$

Specifically, we use soft updates [35] for the target Q-network, i.e., $\bar{\theta} \leftarrow \rho\bar{\theta} + (1-\rho)\theta$, where $\rho \in (0,1)$ is the interpolation factor for the target Q-network.

3.5 Beam Search and Large Neighborhood Search

Beam Search (BS) [50] is an incomplete derivative of Branch-and-Bound (BnB) [33]. Different from the systematic search of BnB, BS selects the partial assignments that satisfy the optimal solution conditions to branch. It uses breadth-first search to construct its search tree. That is, at each level of the search tree, only the predetermined number (called the beam width) of the best partial assignments is selected for further branching, while the remaining partial assignments are permanently pruned. The beam width determines the maximal amount of memory to perform BS. The larger the beam width, the fewer the partial assignments pruned. When the beam width is infinite, no partial assignment is pruned, and thus BS is the same as breadth-first search.

Large Neighborhood Search (LNS) [51] is a meta-heuristic that iteratively explores complex neighborhoods in a search space to find better candidate solutions. The algorithm iteratively improves an initial solution by repeatedly performing destroy and repair phases. Specifically, a subset of variables (i.e., destroyed variables) is selected to discard their current values in the destroy phase. During the repair phase, new assignments are found for the destroyed variables, provided the undestroyed variables retain their value from the previous iteration. Compared with other local search techniques, LNS is characterized by exploring destroyed variables at each step, with the aims of driving the search process away from local optima and finding better candidate solutions.

4 The Proposed Method

The pipeline of our proposed DRL framework is shown in Fig. 2. We first reformulate a WCSP according to MDP. Then, we give the graph representation and graph embedding of a WCSP instance, i.e., a parametric function that encodes the input problem and outputs the predicted Q-values. Next, we present a DRL training algorithm, which determines how to learn the parameter of the Q-network. Finally, we embed the learned Q-network into two meta-heuristics, including beam search and large neighborhood search.

4.1 The MDP Formulation

Given a variable ordering, the solution to a WCSP instance can be constructed by sequentially extending a partial assignment until a full assignment is reached. Therefore, it is natural to model the solution constructing process as an MDP where the state is the remaining sub-problem to be solved (i.e., the problem P under the partial assignment Γ and x_i as the first variable to be

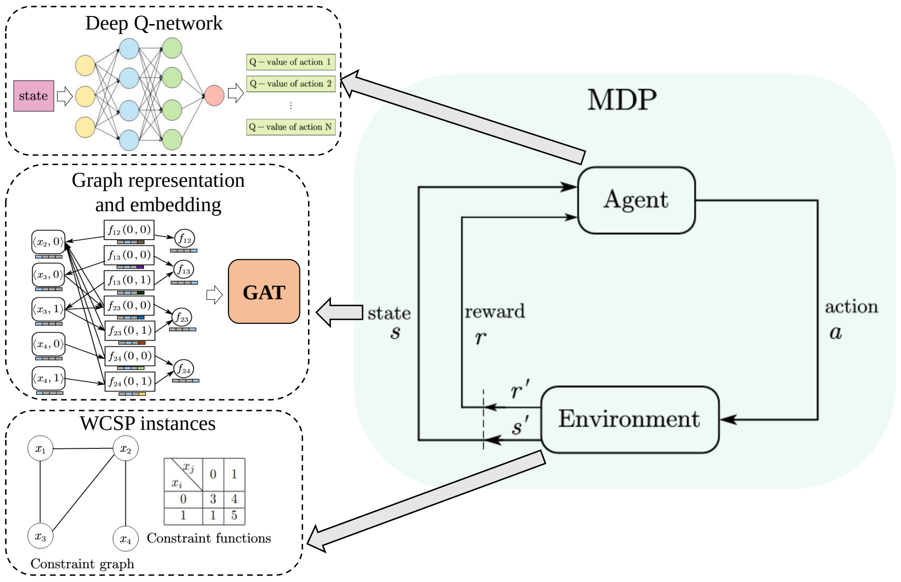


Fig. 2: The pipeline of our proposed method

instantiated in the remaining subproblem. We name that variable as the target variable in the following description), the action space is the domain of x_i and the reward is the change of the current cost incurred by a new assignment of x_i, respectively.

A straightforward approach to order variables is assuming a global ordering over all variables, e.g., an alphabetic ordering. However, such ordering ignores the problem structure and fails to decompose the problem efficiently. Taking Fig. 3(c) as an example, the problem to be solved contains x_3 and x_4. Since there is no constraint between x_3 and x_4, the problems of selecting values for x_3 and x_4 can be solved independently, thereby reducing the computational overheads. Therefore, we propose to use pseudo-trees[1] to sort the variables.

We denote the problem P given the partial assignment Γ and a target variable x_i as $\langle P, \Gamma, x_i \rangle$. Since we use a pseudo tree as the ordering, after assigning x_i with a value $d_i \in D_i$, the problem is reduced to multiple independent subproblems, i.e., each of them rooted at a child of x_i. Therefore, the transition state in our MDP formulation consists of a set of independent sub-states, i.e., $s' = \{s_c | \forall x_c \in C(x_i)\}$ where $s_c = \langle P, \Gamma \cup \{\langle x_i, d_i \rangle\}, x_c \rangle$. As a result, the Bellman Equation in our MDP formulation needs to be modified as:

$$Q_{\pi^*}(s,a) = r(s,a) + \gamma \sum_{x_c \in C(x_i)} \max_{a_c} Q_{\pi^*}(s_c, a_c) \tag{10}$$

[1] We use a depth-first traversal with a max-degree heuristic to build a pseudo tree, where the ties are broken in alphabetical order.

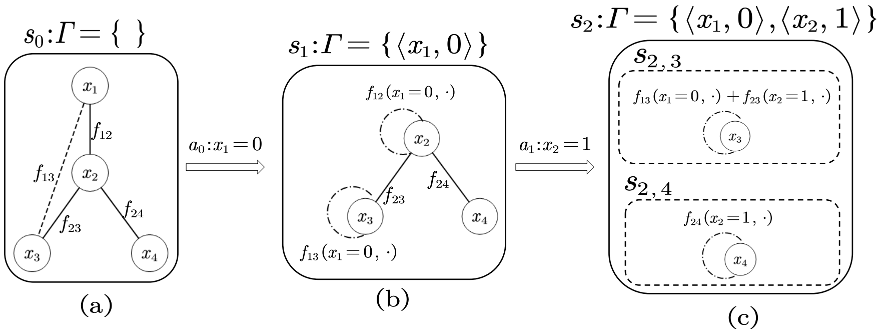


Fig. 3: The solution construction process in WCSP solving as an MDP

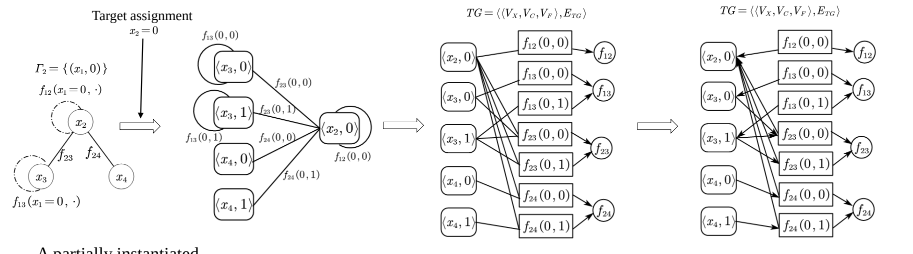
where a_c is the action of x_c. The MDP with multiple independent transition states has been noticed and addressed in [31,30,52], where the modified MDP formulation is used to learn the recursive algorithm selection heuristics in [31], the rules for selecting branches in the Davis-Putnam-Logemann-Loveland (DPLL) procedure in [30] and the variable ordering heuristics for solving CSPs in [52]. Our MDP formulation is the same as the one in [31,30], but differs from that in [52] in the way that the immediate rewards are obtained. In our case and [31,30], the immediate reward is complete, while it is only relevant to a sub-state of the transition state in [52].

Formally, the state, action, transition, and reward in our MDP formulation are defined as follows:

- State: a state s is the WCSP instance P instantiated with Γ and x_i as the target variable. That is, $s = \langle P, \Gamma, x_i \rangle$.
- Action: an action $a = \langle x_i, d_i \rangle$ corresponds to assigning x_i with a value $d_i \in D_i$.
- Transition: transition is deterministic, i.e, $\Pr(s, \langle x_i, d_i \rangle, s') = 1$. Since we use a pseudo tree to order variables, after taking an action $\langle x_i, d_i \rangle$ at the state $s = \langle P, \Gamma, x_i \rangle$, the transition state is $s' = \{s_c | \forall x_c \in C(x_i)\}$ where $s_c = \langle P, \Gamma \cup \{\langle x_i, d_i \rangle\}, x_c \rangle$.
- Reward: the reward function $r(s, \langle x_i, d_i \rangle)$ is defined as the negative increment of the current cost when assigning x_i with d_i at $s = \langle P, \Gamma, x_i \rangle$, i.e.,

$$r(s, \langle x_i, d_i \rangle) = -\sum_{x_j \in AP(x_i)} f_{ij}\left(d_i, \Gamma_{[x_j]}\right) \tag{11}$$

It is worth mentioning that the objective function of our MDP formulation is aligned with that of WCSPs when the discount factor $\gamma = 1$. That is because the negative optimal Q-value of $s = \langle P, \Gamma, x_i \rangle$ is equal to the optimal cost of the sub-problem rooted at $x_i, \forall x_i \in X$. When x_i is the root variable, the negative optimal Q-value of s is equal to the cost of an optimal solution to the problem.



(a) A partially instantiated WCSP instance (b) The microstructure (c) The tripartite graph (d) The directed tripartite graph

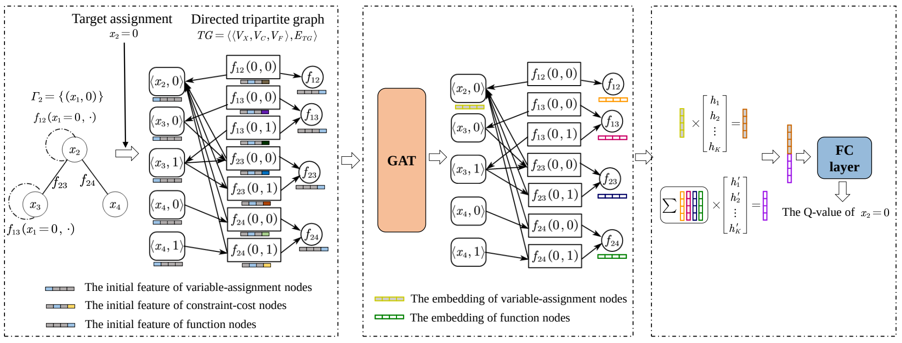
Fig. 4: The graph representation of a partially instantiated WCSP

4.2 Graph Representation and Graph Embedding

A critical step of applying DL to solve combinatorial optimization problems is to build an appropriate structure. Selsam et al. [49] proposed to encode a SAT as an unweighted bipartite graph, where each literal and clause corresponds to a node, and the association between the literal and clause corresponds to an edge. However, the problem structure of WCSPs is quite different from that of SAT in many ways. For instance, the constraint costs of WCSPs are arbitrary positive real values, which cannot be represented by an unweighted bipartite graph. Furthermore, a variable in a WCSP may have more than two values in its domain, while variables in a SAT are restricted to a boolean domain. Xu et al. [60] proposed to represent CSPs as a matrix form, where each entry indicates whether the corresponding assignment is allowed or not. Unfortunately, the representation scheme cannot scale to arbitrary problem sizes. Recently, Song et al. [52] proposed to represent the underlying constraint network of a CSP as GNNs, with variables as vertices and constraints as edges. This method also cannot explicitly handle the cost values in each constraint, making it unsuitable for WCSPs. Thus, we opt for representing a WCSP as a directed and acyclic tripartite graph according to [16].

The generation of the graph representation consists of the following three steps (cf. Fig. 4): first, the partially instantiated WCSP instances are converted into a micro-structure representation [27]; then the micro-structure is compiled into a tripartite graph; finally, the tripartite graph is transformed to a directed acyclic graph. The constraint graph of a WCSP instance $P = (X, D, F)$ given in Fig. 1 under the partial assignment $\Gamma = \{(x_1, 0)\}$ is shown in Fig. 4(a), where the dot lines represents the unary functions. At the first step, the constraint graph is converted into a micro-structure representation, a weighted undirected graph as shown in Fig. 4(b), where each variable assignment that is compatible with Γ corresponds to a vertex, and the constraint cost between a pair of vertices is represented by a weighted edge.

Afterward, the micro-structure is compiled into a tripartite graph $TG = \langle (V_X, V_C, V_F), E_{TG}\rangle$ (cf. Fig. 4(c)) where V_X and V_C correspond to variable assignment nodes and constraint costs (i.e., weighted edges) in the micro-



(a) Graph representation (b) Graph embedding (c) Prediction

Fig. 5: The graph representation and embeddings of a partially instantiated WCSP

structure, respectively, and V_F corresponds to the constraint function in the WCSP instance. Among them, V_C is introduced to explicitly embed the tabular cost structure of a WCSP. Such cost nodes are then connected to the respected assignment nodes (i.e., edges between cost nodes and assignment nodes in E_{TG}) to reflect constraint relationships. V_F is introduced to learn the representation of each function node by summarizing the embedding of the corresponding cost nodes via the directed edges from cost nodes to the function nodes in E_{TG}.

As noted in [34,16], the loopy nature of undirected micro-structure may cause an over-smoothing issue. Thus, we make the tripartite graph directed and acyclic by determining the directions between constraint cost nodes and variable assignment nodes (cf. Fig. 4(d)) through the following two stages. At first, the constraint graph induced by the set of unassigned variables is constructed as a pseudo tree rooted at the target variable. Then, for each constrained variable pair x_i and x_j with $x_i \in AP(x_j)$ and their assignments (i.e., $\langle x_i, d_i\rangle$ and $\langle x_j, d_j\rangle$), the variable-assignment node of $\langle x_i, d_i\rangle$ is set to be the precursor of the constraint-cost node of $f_{ij}(d_i, d_j)$ and the node of $f_{ij}(d_i, d_j)$ is set to be the precursor of the node of $\langle x_j, d_j\rangle$.

Since a partially instantiated WCSP instance is represented as a directed tripartite graph, we use GATs to construct a model that learns graph embeddings (cf. Fig. 5(a) and (b)). In the GAT model, we set a four-dimensional initial feature vector $h_i^{(0)}$ for each node i. For the feature vector $h_i^{(0)}$, its first three elements represent the type of node i (i.e., variable assignment node, constraint cost node, and function node) and its last element is set to be the constraint cost of node i if it is a constraint-cost, and 0 otherwise. Subsequently, $h_i^{(0)}$ is embedded through T layers of the GAT by:

$$h_i^{(t)} = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in N_i}\exp\left(\alpha_{ij}^{k}\mathbf{W}^{k,(t)}h_j^{(t-1)}\right)\right), \forall t = 1, 2, ..., T \qquad (12)$$

where $h_i^{(t)}$ is the embeddings of node i in the t-th time step and $\{\mathbf{W}^{k,(t)} | \forall k = 1, 2, ..., K\}$ is the parameter of the t-th layer of the GAT model.

Given a target variable assignment $\langle x_i, d_i \rangle$ and a partial assignment Γ, the optimal cost of the problem P instantiated with $\Gamma \cup \{\langle x_i, d_i \rangle\}$ can be predicted according to the embeddings of the variable-assignment node i (i.e., $\langle x_i, d_i \rangle$) and the cumulative embeddings of all function nodes in the tripartite graph (cf. Fig. 5(c)). That is,

$$\hat{c}_i = W_3 \left(\text{Concat} \left(W_1 h_i^{(T)}, W_2 \sum_{v_j \in V_F} h_j^{(T)} \right) \right) + b_1 \tag{13}$$

where W_1, W_2, W_3 and b_1 are the parameters of three 1-layer Multi-layer Perceptrons (MLPs). For the sake of description, we denote the GAT model trained by our DRL framework as Q_θ.

Algorithm 1 Deep Q-network for training the model

```
Require: : number of training epochs $N$, number of training iterations $K$, problem distribution $\mathcal{P}$ and experience replay memory $\mathcal{B}$ to capacity $N$
 1: for $n = 1, ..., N$ do
 2:     $P \equiv \langle X, D, F \rangle \sim \mathcal{P}$, $\Gamma \leftarrow \emptyset$
 3:     build a pseudo tree for $P$ rooted at $x_i$
 4:     DFSDECISION($\Gamma, x_i$)
 5:     for $k = 1, ..., K$ do
 6:         $B_s \leftarrow$ sample random mini-batch of data from $\mathcal{B}$
 7:         train the model $Q_\theta$ to minimize Eq. (14)
 8:         $\bar{\theta} \leftarrow \rho\bar{\theta} + (1 - \rho)\theta$
 9: function DFSDECISION($\Gamma, x_i$)
10:     $s \leftarrow \langle P, \Gamma, x_i \rangle$
11:     $d_i = \begin{cases} \text{random value } d'_i \in D_i, \text{w.p. } \epsilon \\ \arg\max_{d'_i \in D_i} Q_\theta\left(s, \langle x_i, d'_i \rangle\right), \text{otherwise} \end{cases}$
12:     compute $r$ by Eq. (11), $\Gamma \leftarrow \Gamma \cup \{\langle x_i, d_i \rangle\}$
13:     $s' \leftarrow \{\langle P, \Gamma, x_c \rangle | \forall x_c \in C(x_i)\}$, $\mathcal{B} \leftarrow \mathcal{B} \cup \{(s, \langle x_i, d_i \rangle, r, s')\}$
14:     for $x_c \in C(x_i)$ do
15:         DFSDECISION($\Gamma, x_c$)
```

4.3 The DRL Training Algorithm

To learn the parameter of our DRL model, we use Deep Q-network (DQN) as illustrated in Algo. 1. In the algorithm, the experiences are generated and stored into a capacitated experience replay memory $\mathcal{B}$ (line 1-4, 9-15). Afterward, the DRL model is trained with the data from the experience memory (line 5-8). Specifically, DQN first samples a WCSP instance from the problem distribution and builds a pseudo tree to order variables (line 2-3). Then, it progressively chooses values for all variables to generate experiences (line 4, 9-15). For each variable x_i given the partial assignment Γ, DQN calls Function **DFSDECISION** to assign x_i based on the ϵ-greedy policy, compute the reward and transition state (line 10-12). The experience, including the initial state, action, reward, and transition state, is then added to the experience memory (line 13). Subsequently, DQN calls Function **DFSDECISION** to select values for the children of x_i (line 14-15). After all variables have been assigned, DQN uniformly samples a batch of data from the experience memory, and train the DRL model with the temporal difference error (line 6-7). That is,

$$\mathcal{L}(\theta) = \frac{1}{|B_s|} \sum_{\langle s,\langle x_i,d_i\rangle,r,s'\rangle \in B_s} \left(Q_\theta(s,\langle x_i,d_i\rangle) - y(s')\right)^2 \tag{14}$$

where $s' = \{s_c | \forall x_c \in C(x_i)\}$ and $y(s') = r + \gamma \sum_{x_c \in C(x_i)} \max_{d_c \in D_c} Q_{\bar{\theta}}(s_c, \langle x_c, d_c\rangle)$. Lastly, the target Q-network is updated using the latest parameter values of the main Q-network and the interpolation factor ρ (line 8).

Algorithm 2 Beam search with our DRL model

```
Require: : a WCSP instance $P$, the DRL model $Q_\theta$, the parameters $k_{bw}$ and $k_{ext}$
 1: $PT \leftarrow$ build a pseudo tree for $P$
 2: $B \leftarrow \{\emptyset | \forall i = \{1, ..., k_{bw}\}\}$
 3: for $l = 1, ..., l^*_{PT}$ do
 4:     for $x_i \in X, \text{level}_{PT}(x_i) = l$ do
 5:         $B' \leftarrow \emptyset$, $E' \leftarrow \emptyset$
 6:         for $j = 1, ..., k_{bw}$ do
 7:             $\Gamma \leftarrow B_{[j]}$, $E \leftarrow \{q(\Gamma, \langle x_i, d_i\rangle)$ computed by Eq. (15) $| \forall d_i \in D_i\}$
 8:             $Ind \leftarrow$the index of the $k_{ext}$ smallest elements in $E$
 9:             $D'_i \leftarrow \{d_i | \forall d_i \in D_i, i \in Ind\}$
10:             $B' \leftarrow B' \cup \left\{\langle \Gamma \cup \{\langle x_i, d_i\rangle\} | \forall d_i \in D'_i\right\}$
11:             $E' \leftarrow E' \cup \{cost(\Gamma) + q(\Gamma, \langle x_i, d_i\rangle) | \forall d_i \in D'_i\}$
12:         $Ind' \leftarrow$the index of the $k_{bw}$ smallest elements in $E'$
13:         $B \leftarrow \left\{B'_{[i]} | \forall i \in Ind'\right\}$
14: return $\arg\min_{\Gamma \in B} cost(\Gamma)$
```

4.4 Heuristics for Beam Search and Large Neighborhood Search

Besides constructing solutions greedily from scratch, in this section, we show that our DRL model can also be embedded into various meta-heuristics. In particular, we consider two well-known meta-heuristics, i.e., Beam Search (BS) and Large Neighborhood Search (LNS) and show how to apply the predicted Q-values to these heuristics.

Since the Q-values in our model approximate the optimal cost of the subproblem rooted at a variable, one can use the predicted Q-values as a proxy to evaluate the quality of each assignment. Thus, our DRL model can be embedded into BS to guide the expansion of partial assignments and determine

Table 1: The intermediate results of x_2 when executing BS

Γ	$q(\Gamma, \langle x_2, 0\rangle)$	$q(\Gamma, \langle x_2, 1\rangle)$	d_2	$cost(\Gamma)$	B'	E'
$\{\langle x_1, 0\rangle\}$	9	11	0	0	$\{\{\langle x_1, 0\rangle, \langle x_2, 0\rangle\},$ $\{\langle x_1, 1\rangle, \langle x_2, 0\rangle\}\}$	$\{9, 10\}$
$\{\langle x_1, 1\rangle\}$	10	11	0	0		

which extended partial assignments are eligible for further branching. Algo. 2 gives the pseudo-code of BS with our DRL model, where the beam B is introduced to track the most promising partial assignments and initialized as k_{wb} (called the beam width) empty sets at the beginning of the algorithm (line 2). BS first orders variables by using a pseudo tree PT (line 1) where l^*_{PT} is the maximal level of variables in PT (line 3) and $\text{level}_{PT}(x_i)$ is the level of x_i in PT (line 4). Then, it explores the search tree according to the level of each variable in PT (lines 3-13). For x_i, a variable at the l-th level of PT, BS works by extending each partial assignment $\Gamma \in B$ at most k_{ext} possible ways according to their quality (line 6-9). The quality of each value $d_i \in D_i$ under Γ is evaluated by:

$$q(\Gamma, \langle x_i, d_i\rangle) = -Q_\theta(\langle P, \Gamma, x_i\rangle, \langle x_i, d_i\rangle) \tag{15}$$

The extended partial assignments and their evaluation values are then stored in a new beam B' and E', respectively (line 10-11). After all partial assignments in B have been processed, the algorithm updates B with the top k_{bw} optimal partial assignments in B' (line 12-13).

Next, we will take the variable x_2 in Fig. 1(b) as an example to show how our DRL model is embedded into BS with $k_{wb} = 2$ and $k_{ext} = 1$. Assuming that $B = \{\{\langle x_1, 0\rangle\}, \{\langle x_1, 1\rangle\}\}$ and $D_2 = \{0, 1\}$, x_2 expands each partial assignment $\Gamma \in B$ with its corresponding best value $d_2 \in D_2$ and the intermediate results can be found in Table 1. For $\Gamma = B_{[1]} = \{\langle x_1, 0\rangle\}$, it computes the quality evaluation for each value in D_2 given Γ by Eq. (15). Thus, we have $q(\Gamma, \langle x_2, 0\rangle) = 9$ and $q(\Gamma, \langle x_2, 1\rangle) = 11$ as shown in the first row of Table 1. Since $q(\Gamma, \langle x_2, 0\rangle) < q(\Gamma, \langle x_2, 1\rangle)$ and $k_{ext} = 1$, x_2 assigns itself with 0 and extends Γ with its assignment, i.e., the extended assignment is $\Gamma' = \Gamma \cup \{\langle x_2, 0\rangle\}$. Then, it adds Γ' and the corresponding evaluation value (i.e., $cost(\Gamma) + q(\Gamma, \langle x_2, 0\rangle) = 9$) to B' and E', respectively. Thus, we have $B' = \{\langle x_1, 0\rangle, \langle x_2, 0\rangle\}$ and $E' = \{9\}$.

For $\Gamma = B_{[2]} = \{\langle x_1, 1\rangle\}$, x_2 performs a similar procedure as above. As shown in the second row of Table 1, it selects 0 for itself (since $q(\Gamma, \langle x_2, 0\rangle) < q(\Gamma, \langle x_2, 1\rangle)$) and expands Γ to $\Gamma' = \Gamma \cup \{\langle x_2, 0\rangle\}$. Afterward, Γ' and the corresponding evaluation value are added to B' and E', respectively. Now, $B' = \{\{\langle x_1, 0\rangle, \langle x_2, 0\rangle\}, \{\langle x_1, 1\rangle, \langle x_2, 0\rangle\}\}$ and $E' = \{9, 10\}$. Since $|B'| = k_{wb} = 2$, no partial assignment in B' will be abandoned and thus the beam for x_3 is $B = B' = \{\{\langle x_1, 0\rangle, \langle x_2, 0\rangle\}, \{\langle x_1, 1\rangle, \langle x_2, 0\rangle\}\}$.

Our model can also be embedded into LNS as a repair policy which solves each connected sub-problem by assigning variables according to the predicted Q-values. The pseudo-code of LNS with our DRL model can be found in Algo.

Algorithm 3 Large neighborhood search with our DRL model

```
Require: : a WCSP instance $P$, the DRL model $Q_\theta$, the proportion of destroyed variables
    $p$, number of iterations $T$
 1: $sol \leftarrow$ a random solution
 2: for t=1,...,T do
    Destroy phase
 3:     $X_{des} \leftarrow$ uniformly select $|X|p$ variables from $X$
    Repair phase
 4:     $\Gamma \leftarrow \{\langle x_i, sol_{[x_i]}\rangle | \forall x_i \notin X_{des}\}$
 5:     for all connected sub-problem $X_{cr} \subseteq X_{des}$ do
 6:         $PT \leftarrow$ build a pseudo tree for $X_{cr}$
 7:         for $l = 1, ..., l^*_{PT}$ do
 8:             for $x_i \in X, \text{level}_{PT}(x_i) = l$ do
 9:                 compute $d^*_i$ by Eq. (16), $\Gamma \leftarrow \Gamma \cup \{\langle x_i, d^*_i\rangle\}$
10:     update $sol$ with $\Gamma$
```

3. The algorithm iteratively improves an initial solution by repeatedly executing destroy and repair phases (line 1-10). During the destroy phase, LNS selects the Large Neighborhood (LN) variables whose current assignment will be discarded (line 3). Then, in the repair phase, LNS finds a new assignment for destroyed variables given the assignment of undestroyed variables from the previous iteration (line 4-10). The assignment for a destroyed variable x_i is computed by:

$$d^*_i = \arg\max_{d_i \in D_i} Q_\theta(\langle P, \Gamma, x_i\rangle, \langle x_i, d_i\rangle) \tag{16}$$

5 Experimental Evaluations

In this section, we conduct extensive empirical studies. We first present the details of the experiments and training stage. Then, we compare the heuristics obtained from our DRL model[2] against those derived from the SL model to illustrate the superiority of our DRL framework. Finally, we compare our DRL-based algorithms with their counterparts with traditional tabular-based heuristics (i.e., mini-bucket elimination, MBE) and state-of-the-art incomplete WCSP algorithms on various standard benchmarks, with the aims of showing the capability of our DRL model to boost WCSP algorithms.

5.1 Benchmarks

In this experiment, the algorithms are benchmarked on four types of problems, including random WCSPs, weighted graph coloring problems, scale-free networks and random meeting scheduling problems.

– *Random WCSPs* are a general form for WCSPs, where a set of variables are randomly constrained with one another. In the experiments, we set

[2] Our DRL model is available at https://github.com/czy920/DRL4WCSP.

the number of variables to 70 or 120, domain size to 10, and each constraint uniformly chosen from [0, 100], and consider the problems with graph density 0.1 for the sparse configuration and 0.6 for the dense configuration. Besides, we also evaluate the performance of the algorithms on random WCSPs with the graph density of 0.05, the domain size of 3, and the number of variables varying from 100 to 300
- *Weighted graph coloring problems* are the problems in which every vertex should be colored, and two adjacent vertexes should have different colors. In the experiments, we consider weighted graph coloring problems with 3 available colors for each variable, the graph density of 0.05, and varying the number of variables from 100 to 300. The cost of each violation is chosen uniformly from 0 to 100.
- *Scale-free networks* [2] are networks whose degree distribution follows a power law. The experiment uses Barabási-Albert (BA) model to generate the constraint graph topology. In the beginning, we set the number of initial variables to 10 and connect them randomly. In an iteration of the BA process, we add a new variable and connect it with 3 variables (for sparse problems) or 10 variables (for dense problems) with a probability proportional to the current number of links. We set the variable number to 120, and the other parameters are the same as that of random WCSPs.
- *Random meeting scheduling problems* [64] are problems of persons scheduling a set of meetings. We randomly select a travel time for each pair of meetings. When the difference between the time-slots of two meetings with overlapping persons is less than the travel time, the persons in both meetings are considered overbooked, and the cost is defined as the number of the overbooked persons. In the experiments, we set the number of participants to 90, the number of meetings to 20, and the available time-slots to 20. Each person randomly participates in two meetings, and travel times are uniformly selected from 6 to 10.

5.2 Baselines

To demonstrate the superiority of the proposed DRL framework, we compared the DRL-boosted algorithms with three types of baselines: meta-heuristics for combinational optimization problems, belief propagation and local search.

- *Meta-heuristics.* We consider three meta-heuristics, including Greedy Search (GS), Beam Search (BS) and Large Neighborhood Search (LNS), to combine with the heuristics derived from our DRL model, those obtained from the SL model [16] and traditional tabular-based heuristics (i.e., mini-bucket elimination, MBE [15]). As for the hyper-parameter of traditional tabular-based heuristics (i.e., MBE) and two meta-heuristics (i.e., BS and LNS), we set the memory budget of MBE to d^6, $k_{wb} = 4$ and $k_{ext} = 2$ for BS and the destroy probability to 0.2 for LNS according to the parameter setting in [16]. Additionally, we consider the traditional tabular-based LNS algorithm (i.e., T-LNS [26]) with the destroy probability to 0.5. In T-LNS, bucket elimination is used to solve tree-structured relaxation of the sub-problem induced by the destroyed variables.

– *Belief propagation algorithms.* Max-Sum [19] is a widely used belief propagation algorithm, which has been applied to many real-world applications including the wide area surveillance problem [18] and the smart environment configuration problem [45]. In the experiment, we consider one of the strongest variant of Max-sum, i.e., Damped Max-Sum (DMS) [11] with damping factor of 0.9.
– *Local search algorithms.* We consider a Stochastic Algorithm (SA) [63] with the parallel probability to 0.8 and GDBA [39] with $\langle M, NM, T\rangle$. SA is chosen since it is the classical local search algorithm, while GDBA is chosen since it is the state-of-the-art local search algorithm.

In the experiment, we evaluate the solution quality of all the algorithms with the metric of *normalized anytime cost*, which is the cost of the best solution found in each run normalized by the number of constraints. We set the timeout for each algorithm to 45 minutes. The algorithm terminates when it converges or exceeds the time limit. For a fair comparison, the configuration of the GAT model we use is the same as that in [16]. Specifically, our model consists of 4 GAT layers (i.e. $T = 4$), where the first 3 layers have 8 output channels and 8 attention heads, and the latter layer has 16 outputs channel and 4 attention heads. Besides, we use ELU [10] as the activation function for each GAT layer. The training set and validation set in the experiment are derived from a WCSP random distribution with $|X| \in [40, 60]$, $d \in [3, 15]$, $p_1 \in [0.1, 0.4]$ where p_1 is the graph density of the WCSP instance. The GAT model is implemented with the PyTorch geometry framework [20] and trained with the Adam optimizer [29] with a learning rate of 0.0001 and a weight decay rate of 0.00001. For the hyper-parameters of the deep Q-network algorithm, we set the discount factor γ to 0.99, the batch size to 64, and the number of training steps to 15000. Finally, we average the experimental results over 50 independently generated problems and evaluate the solution quality of all algorithms after running for the same wall-clock time. All experiments are carried out on an Intel i7-7820X workstation with GeForce RTX 3090 GPUs.

5.3 Performance Comparisons

5.3.1 Ablation Study

Fig. 6 shows the comparison of the DRL-based and SL-based algorithms on random WCSPs with different number of variables. It can be seen from the figure that the DRL-based algorithms have great advantages over the SL-based algorithms on all the problems. Specifically, GS_DRL, BS_DRL and LNS_DRL outperform GS_SL, BS_SL, LNS_SL by about 0.71∼5.51%, 0.94∼3.11% and 0.81∼3.63%, respectively. This phenomenon shows that the heuristics derived from our DRL model is better than those obtained from the SL model, and the gap become wider when solving large-scale problems. That is because training

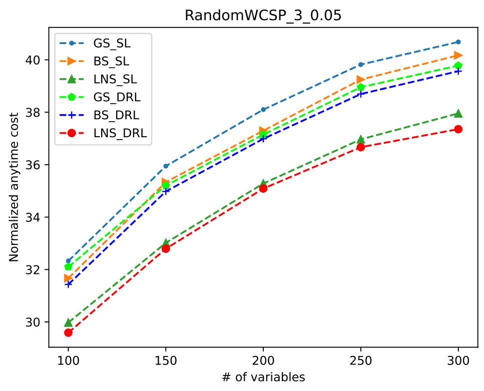
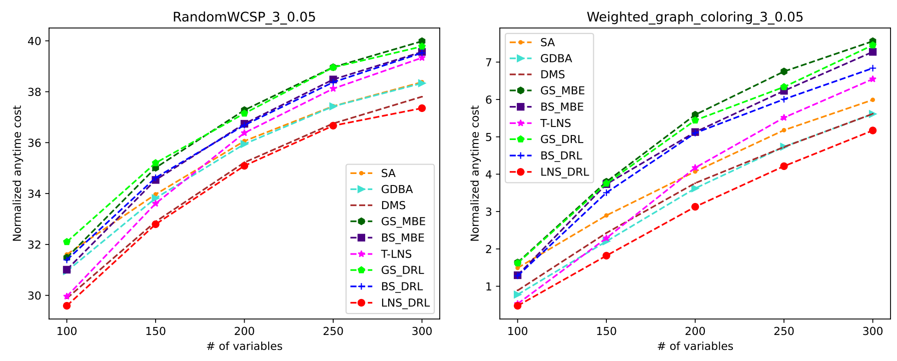


Fig. 6: Solution quality comparisons of the heuristics derived from our DRL model and the SL model on random WCSPs

on large-scale problems helps the model to mine more sophisticated patterns of WCSPs, and the scale of training problems for DRL is much larger than that for SL. Specifically, SL uses optimal labeled data to train the model. However, obtaining optimal labeled data is quite challenging, making SL only trainable on small-scale problems. In the experiment, the training data for SL comes from a random WCSP distribution with $|X| \in [15, 30]$, $d \in [3, 15]$ and $p_1 \in [0.1, 0.4]$. The DRL training process can be seen as an alternation between finding “high-quality labeled data” and performing SL on the collected data. Thus, DRL can train models on large-scale problems. The training problems for DRL comes from a random WCSP distribution with $|X| \in [40, 60]$, $d \in [3, 15]$ and $p_1 \in [0.1, 0.4]$.

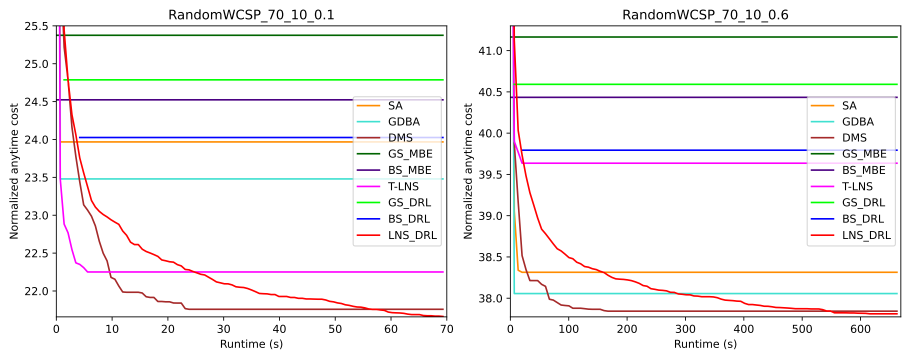
5.3.2 Results on random WCSPs

Fig. 7 shows solution quality results for all the baselines on random WCSPs and Weighted Graph Coloring (WGC) problems. Here, the results of SL-boosted algorithms [16] are not concluded in Fig. 7-10, since these algorithms are inferior to our DRL-boosted algorithms on benchmark problems (as shown in Fig. 12-16 in the appendix), and there are many comparing baselines. It can be seen from Fig. 7, the performance of the DRL-based algorithms on small-scale problems (e.g., $|X|$=100) is similar to their counterparts with MBE. The advantage of our DRL-based algorithms comes out as the number of variables increases. Specifically, our DRL-based algorithms improve their counterparts with MBE by about 3.37∼6.23% on random WCSPs with 300 variables and 1.17∼25.4% on WGC problems with 300 variables. That is because the induced width of the built pseudo tree increases with the number of variables. Concretely, the average induced width is 43 for the problems with 100 variables and 223 for the problems with 300 variables. Therefore, in the face of complex problems, MBE needs to approximate more dimensions in the variable elim-



(a) (b)

Fig. 7: Solution quality comparisons: (a) random WCSPs; (b) weighted graph coloring problems

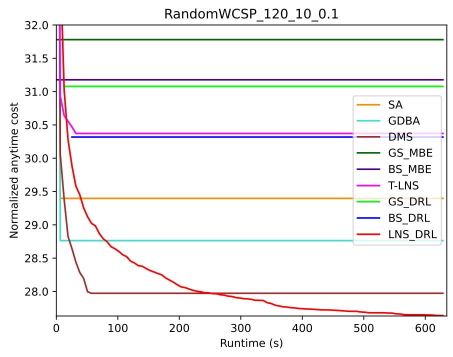


(a) (b)

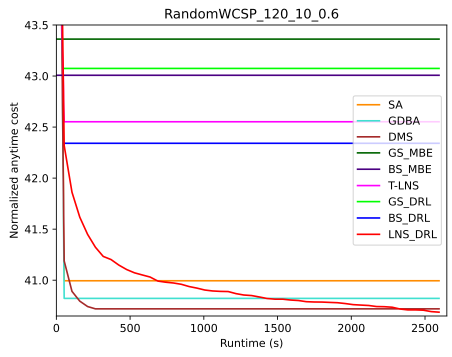
Fig. 8: Solution quality on random WCSPs (70 variables)

ination process and cannot provide tight enough bounds. T-LNS attempts to optimize by optimally solving tree-structure relaxations of sub-problems induced by destroyed variables in each round. It also performs poorly when solving large-scale problems (e.g., problems with more than 200 variables) due to ignoring many constraints. In contrast, our LNS_DRL solves the induced problem without relaxation, which is a significant improvement over state-of-the-art methods such as DMS. Precisely, LNS_DRL outperforms DMS by about 1.12∼2.07% on random WCSPs and 7.93∼25.03% on WGC problems.

Fig. 8 shows the comparison results of random WCSPs with the number of variables of 70. It can be seen from the figure that the DRL-based algorithms have obvious advantages over their counterparts with MBE on these problems. Precisely, GS_DRL and BS_DRL excel GS_MBE and BS_MBE by about 2.32% and 2.03% on the sparse problems, respectively, and 1.39% and 1.58% on the dense problems, respectively. LNS_DRL outperforms T-LNS by about 2.64%



(a)

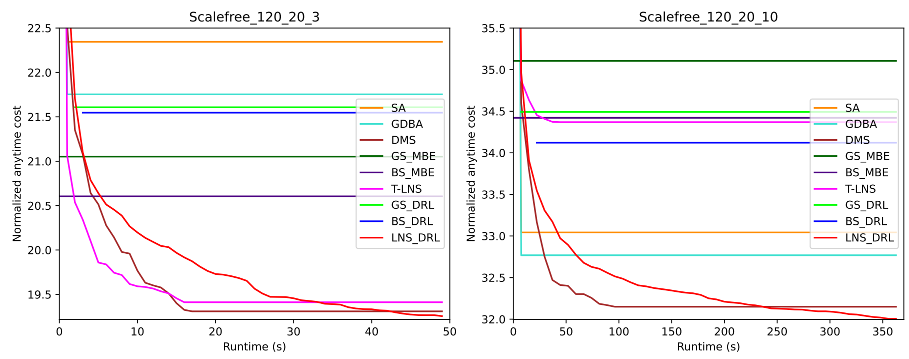


(b)

Fig. 9: Solution quality on random WCSPs (120 variables)

on the sparse problems and 4.6% on the dense problems. These phenomena show that our heuristics can effectively improve the performance of the original algorithms. In addition, it can be seen that the local search algorithms (e.g., SA and GDBA) quickly converge to poor local optima, while DMS can find a better solution in 25 seconds for the sparse problems and 200 seconds for the dense problems. In contrast, our LNS_DRL improves more steadily, outperforming all the baselines after 60 seconds on the sparse problems and 900 seconds on the dense problems. Specifically, LNS_DRL outperforms state-of-the-art algorithms (e.g., DMS) by about 0.43% on the sparse problems and 0.11% on the dense problems. Additionally, we can conclude from the figure that DMS is the best performing algorithm under the limited time budget, but it quickly gets into a local optimum. In contrast, our LNS_DRL can continuously improve the quality of the solution and find a better solution than DMS produces given enough time.

Fig. 9 presents the comparisons on the random WCSPs with 120 variables. Similarly, the DRL-based algorithms have great superiorities over their counterparts with MBE. Concretely, GS_DRL and BS_DRL surpass GS_MBE and BS_MBE by about 2.21% and 2.76% on the sparse problems, respectively, and 0.66% and 1.55% on the dense problems, respectively. LNS_DRL outperforms T-LNS by about 9.01% on the sparse problems and 4.38% on the dense problems. Different from the experimental results shown in Fig. 8, we can see from Fig. 9 that our BS_DRL outperforms T-LNS by about 1.8% on the sparse problems, and the advantage on the dense problems extends to 5.12%. That is because the proportion of constraints ignored by T-LNS when solving the problems with 120 variables is much larger than when solving the problems with 70 variables.



(a) (b)

Fig. 10: Solution quality on scale-free problems

5.3.3 Results on Structured Problems

Fig. 10 shows the comparisons of normalized anytime costs on the sparse and dense scale-free networks, respectively. In the experiment, the average induced width is 32 for the sparse problems and 77 for the dense problems. Unlike the experiment results on random WCSPs, GS_DRL and BS_DRL perform worse than their counterparts with MBE on the sparse scale-free networks. That is because the induced width of sparse problems is relatively small compared to that of random WCSPs, so MBE can provide more efficient bounds for greedy search and beam search. Also, it can be seen from Fig. 10(a) that T-LNS exhibits great superiorities over local search algorithms (i.e., SA and GDBA) and is slightly lower than DMS. However, due to the effectiveness of our heuristics, LNS_DRL still outperforms state-of-arts methods (e.g., DMS) by about 2.48% on the sparse problems. On the dense problems, the advantage of our heuristics comes out in greedy search and beam search. Specifically, GS_DRL and BS_DRL outperform GS_MBE and BS_MBE by about 1.75% and 0.87%, respectively. Although T-LNS exhibits its huge advantage on the sparse problems, its performance on the dense problems is even 0.72% worse than BS_DRL and 6.88% worse than LNS_DRL. These results demonstrate the necessity of our heuristics for LNS to solve structured problems, especially when facing large-scale problems.

Fig. 11 presents the comparisons of normalized anytime costs on the random meeting scheduling problems. The average induced width for this experiment is 14, which is much smaller than that of random WCSPs. Thus, it can be seen that the performance of GS_DRL and BS_DRL is also inferior to their counterparts with MBE, similar to the experimental results on the sparse scale-free networks. Furthermore, T-LNS achieves better performance than all the algorithms expected LNS_DRL. That is because the variables in the problem are under-constrained, and T-LNS only needs to drop a few edges to obtain a tree-structured problem. However, our LNS_DRL still exhibits great superi-

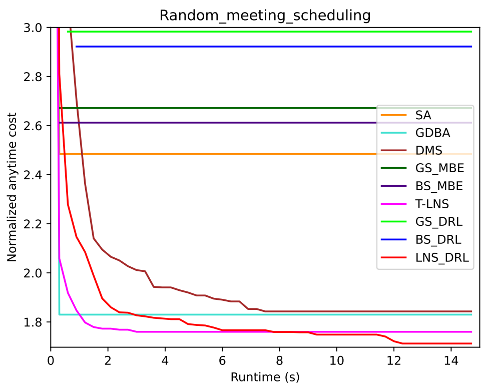


Fig. 11: Solution quality on random meeting scheduling problems

orities over other competitors. Specifically, LNS_DRL is superior to DMS by about 7.87%, GDBA by about 7.05% and T-LNS by about 3.53%, respectively. The results demonstrate that our heuristics can improve the performance of LNS when solving real-world problems.

6 Conclusion

In this paper, we propose a DRL framework that enables the model to be trained on large-scale problems. As a result, the learned model could capture more sophisticated patterns of the problems to generate effective solution construction heuristics for WCSPs. In the framework, we propose to effectively decompose the problem by using a pseudo tree, and formulate the solution construction process as an MDP with multiple independent transition states. Through a modified Bellman Equation, we use GATs parameterized deep Q-value network to learn the optimal Q-values, and the solution construction heuristics are extracted from the learned Q-value network. In addition to constructing greedy solutions, we embed our heuristics into BS by evaluating the quality of partial assignments, and LNS by finding new assignments for destroyed variables via constructing a solution greedily. The extensive empirical evaluations confirm the effectiveness of our proposed DRL framework.

In the future, we plan to improve our DRL framework in the following aspects: at first, we will exploit curriculum learning [37] to make the training processes more stable; then, we will devote to extending our DRL model to handle the problems with hard and higher-arity constraints; finally, we will explore to embed our model into complete WCSP algorithms, such as extracting the branching heuristics from the model for branch-and-bound algorithms.

References

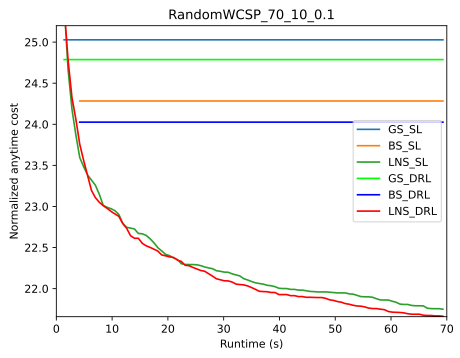
1. Allouche, D., Givry, S.d., Katsirelos, G., Schiex, T., Zytnicki, M.: Anytime hybrid best-first search with tree decomposition for weighted CSP. In: CP, pp. 12–29. Springer (2015)
2. Barabási, A.L., Albert, R.: Emergence of scaling in random networks. Science **286**(5439), 509–512 (1999)
3. Bellman, R.: A markovian decision process. Journal of mathematics and mechanics pp. 679–684 (1957)
4. Bengio, Y., Lodi, A., Prouvost, A.: Machine learning for combinatorial optimization: a methodological tour d'horizon. European Journal of Operational Research **290**(2), 405–421 (2021)
5. Cappart, Q., Moisan, T., Rousseau, L.M., Prémont-Schwarz, I., Cire, A.A.: Combining reinforcement learning and constraint programming for combinatorial optimization. In: AAAI, vol. 35, pp. 3677–3687 (2021)
6. Chalumeau, F., Coulon, I., Cappart, Q., Rousseau, L.M.: Seapearl: A constraint programming solver guided by reinforcement learning. In: CPAIOR, pp. 392–409. Springer (2021)
7. Chen, Z., Deng, Y., Wu, T., He, Z.: 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)
8. Chen, Z., Zhang, W., Deng, Y., Chen, D., Li, Q.: RMB-DPOP: Refining MB-DPOP by reducing redundant inference. In: AAMAS, pp. 249–257 (2020)
9. Cicirello, V.A.: On the design of an adaptive simulated annealing algorithm. In: CP Workshop on Autonomous Search (2007)
10. Clevert, D.A., Unterthiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (ELUS). In: ICLR (2016)
11. Cohen, L., Galiki, R., Zivan, R.: Governing convergence of Max-sum on DCOPs through damping and splitting. Artificial Intelligence **279**, 103212 (2020)
12. De Givry, S., Heras, F., Zytnicki, M., Larrosa, J.: Existential arc consistency: Getting closer to full arc consistency in weighted CSPs. In: IJCAI, vol. 5, pp. 84–89 (2005)
13. Dechter, R.: Bucket elimination: A unifying framework for reasoning. Artificial Intelligence **113**(1-2), 41–85 (1999)
14. Dechter, R., Cohen, D., et al.: Constraint processing. Morgan Kaufmann (2003)
15. Dechter, R., Rish, I.: Mini-buckets: A general scheme for bounded inference. Journal of the ACM (JACM) **50**(2), 107–153 (2003)
16. Deng, Y., Kong, S., An, B.: Pretrained cost model for distributed constraint optimization problems. In: AAAI (2022)
17. Deng, Y., Yu, R., Wang, X., An, B.: Neural regret-matching for distributed constraint optimization problems. In: IJCAI (2021)
18. Farinelli, A., Rogers, A., Jennings, N.R.: Agent-based decentralised coordination for sensor networks using the max-sum algorithm. Autonomous agents and multi-agent systems **28**(3), 337–380 (2014)
19. Farinelli, A., Rogers, A., Petcu, A., Jennings, N.R.: Decentralised coordination of low-power embedded devices using the max-sum algorithm. In: AAMAS, pp. 639–646 (2008)
20. Fey, M., Lenssen, J.E.: Fast graph representation learning with PyTorch geometric. In: ICLR Workshop on Representation Learning on Graphs and Manifolds (2019)
21. Freuder, E.C., Quinn, M.J.: Taking advantage of stable sets of variables in constraint satisfaction problems. In: IJCAI, vol. 85, pp. 1076–1078 (1985)
22. Galassi, A., Lombardi, M., Mello, P., Milano, M.: Model agnostic solution of CSPs via deep learning: A preliminary study. In: CPAIOR, pp. 254–262. Springer (2018)
23. Gaudreault, J., Frayret, J.M., Pesant, G.: Distributed search for supply chain coordination. Computers in Industry **60**(6), 441–451 (2009)
24. Givry, S.d., Lee, J.H., Leung, K.L., Shum, Y.W.: Solving a judge assignment problem using conjunctions of global cost functions. In: CP, pp. 797–812. Springer (2014)
25. Hart, S., Mas-Colell, A.: A simple adaptive procedure leading to correlated equilibrium. Econometrica **68**(5), 1127–1150 (2000)

26. Hoang, K.D., Fioretto, F., Yeoh, W., Pontelli, E., Zivan, R.: A large neighboring search schema for multi-agent optimization. In: CP, pp. 688–706. Springer (2018)
27. Jégou, P.: Decomposition of domains based on the micro-structure of finite constraint-satisfaction problems. In: AAAI, vol. 93, pp. 731–736 (1993)
28. Jiang, Y., Cao, Z., Zhang, J.: Learning to solve 3-D bin packing problem via deep reinforcement learning and constraint programming. IEEE transactions on cybernetics (2021)
29. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: ICLR (2015)
30. Lagoudakis, M.G., Littman, M.L.: Learning to select branching rules in the DPLL procedure for satisfiability. Electronic Notes in Discrete Mathematics **9**, 344–359 (2001)
31. Lagoudakis, M.G., Littman, M.L., et al.: Algorithm selection using reinforcement learning. In: ICML, pp. 511–518. Citeseer (2000)
32. Larrosa, J., Schiex, T.: In the quest of the best form of local consistency for weighted CSP. In: IJCAI, vol. 3, pp. 239–244 (2003)
33. Lawler, E.L., Wood, D.E.: Branch-and-bound methods: A survey. Operations Research **14**(4), 699–719 (1966)
34. Li, G., Muller, M., Thabet, A., Ghanem, B.: DeepGCNs: Can GCNs go as deep as CNNs? In: ICCV, pp. 9267–9276 (2019)
35. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. In: ICLR (2016)
36. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
37. Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M.E., Stone, P.: Curriculum learning for reinforcement learning domains: A framework and survey. Journal of Machine Learning Research **21**, 181:1–181:50 (2020)
38. Nguyen, D.T., Yeoh, W., Lau, H.C., Zivan, R.: Distributed Gibbs: A linear-space sampling-based DCOP algorithm. Journal of Artificial Intelligence Research **64**, 705–748 (2019)
39. Okamoto, S., Zivan, R., Nahon, A., et al.: Distributed breakout: Beyond satisfaction. In: IJCAI, pp. 447–453 (2016)
40. Ottens, B., Dimitrakakis, C., Faltings, B.: DUCT: An upper confidence bound approach to distributed constraint optimization problems. ACM Transactions on Intelligent Systems and Technology **8**(5), 1–27 (2012)
41. Petcu, A., Faltings, B.: DPOP: A scalable method for multiagent constraint optimization. In: IJCAI, pp. 266–271 (2005)
42. Pisinger, D., Ropke, S.: Handbook of Metaheuristics. Springer (2010)
43. Popescu, A., Polat-Erdeniz, S., Felfernig, A., Uta, M., Atas, M., Le, V.M., Pilsl, K., Enzelsberger, M., Tran, T.N.T.: An overview of machine learning techniques in constraint solving. Journal of Intelligent Information Systems pp. 1–28 (2021)
44. Razeghi, Y., Kask, K., Lu, Y., Baldi, P., Agarwal, S., Dechter, R.: Deep bucket elimination. In: IJCAI, pp. 4235–4242 (2021)
45. Rust, P., Picard, G., Ramparany, F.: Using message-passing DCOP algorithms to solve energy-efficient smart environment configuration problems. In: IJCAI, pp. 468–474 (2016)
46. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Transactions on Neural Networks **20**(1), 61–80 (2009)
47. Schiex, T., Fargier, H., Verfaillie, G., et al.: Valued constraint satisfaction problems: Hard and easy problems. In: IJCAI, vol. 95, pp. 631–639 (1995)
48. Selsam, D., Bjørner, N.: Guiding high-performance SAT solvers with unsat-core predictions. In: SAT, pp. 336–353. Springer (2019)
49. Selsam, D., Lamm, M., Benedikt, B., Liang, P., de Moura, L., Dill, D.L., et al.: Learning a SAT solver from single-bit supervision. In: ICLR (2019)
50. Shapiro, S.C.: Encyclopedia of artificial intelligence, second edition. Wiley-Interscience (1992)
51. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: CP, pp. 417–431. Springer (1998)

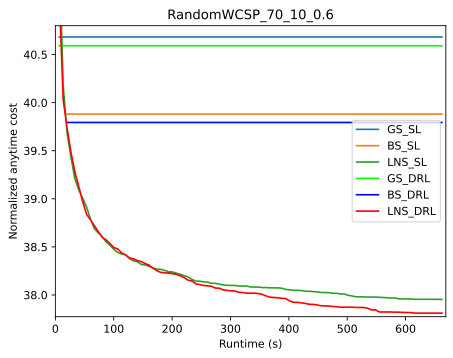
52. Song, W., Cao, Z., Zhang, J., Xu, C., Lim, A.: Learning variable ordering heuristics for solving constraint satisfaction problems. Engineering Applications of Artificial Intelligence **109**, 104603 (2022)
53. Strokach, A., Becerra, D., Corbi-Verge, C., Perez-Riba, A., Kim, P.M.: Fast and flexible protein design using deep graph neural networks. Cell systems **11**(4), 402–411 (2020)
54. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
55. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: NeurIPS, pp. 5998–6008 (2017)
56. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: ICLR (2018)
57. Vinyals, M., Shieh, E., Cerquides, J., Rodriguez-Aguilar, J.A., Yin, Z., Tambe, M., Bowring, E.: Quality guarantees for region optimal DCOP algorithms. In: AAMAS, pp. 133–140 (2011)
58. Vucinic, J., Simoncini, D., Ruffini, M., Barbe, S., Schiex, T.: Positive multistate protein design. Bioinformatics **36**(1), 122–130 (2020)
59. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning **8**(3), 229–256 (1992)
60. Xu, H., Koenig, S., Kumar, T.S.: Towards effective deep learning for constraint satisfaction problems. In: CP, pp. 588–597. Springer (2018)
61. Yolcu, E., Póczos, B.: Learning local search heuristics for boolean satisfiability. In: NeurIPS, pp. 7990–8001 (2019)
62. Zhang, W., Sun, Z., Zhu, Q., Li, G., Cai, S., Xiong, Y., Zhang, L.: NLocalSAT: boosting local search with solution prediction. In: IJCAI, pp. 1177–1183 (2021)
63. Zhang, W., Wang, G., Xing, Z., Wittenburg, L.: Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. Artificial Intelligence **161**(1-2), 55–87 (2005)
64. Zivan, R., Parash, T., Cohen, L., Peled, H., Okamoto, S.: 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)

Appendix

Fig. 12 - 16 present the comparison of the DRL-based and SL-based algorithms on benchmark problems.

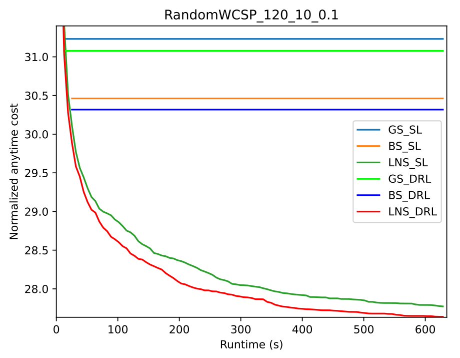


(a)

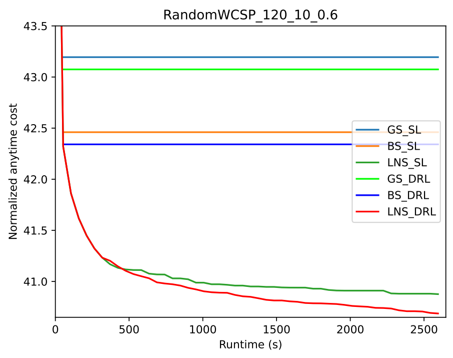


(b)

Fig. 12: Solution quality of the heuristics derived from our DRL model and the SL model on random WCSPs (70 variables)

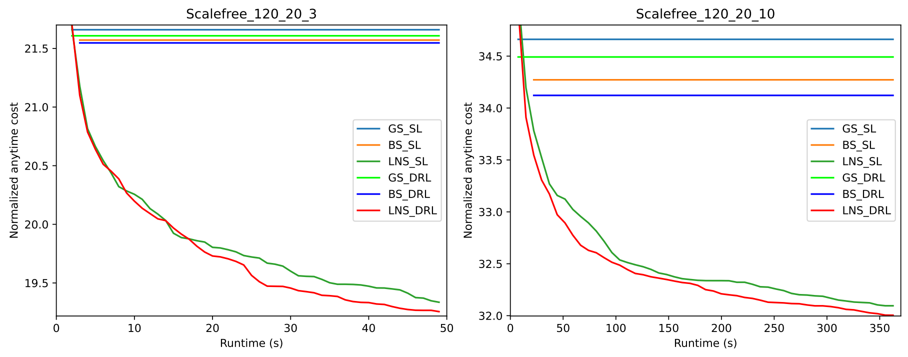


(a)



(b)

Fig. 13: Solution quality of the heuristics derived from our DRL model and the SL model on random WCSPs (120 variables)



(a) (b)

Fig. 14: Solution quality of the heuristics derived from our DRL model and the SL model on scale-free problems

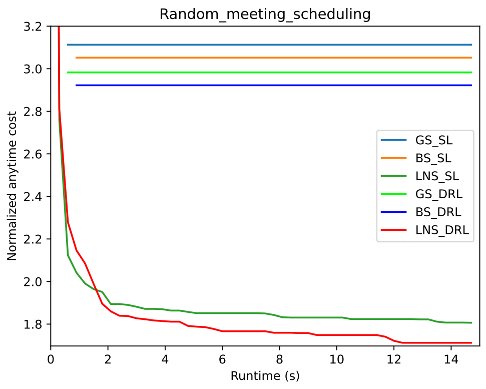


Fig. 15: Solution quality of the heuristics derived from our DRL model and the SL model on random meeting scheduling problems

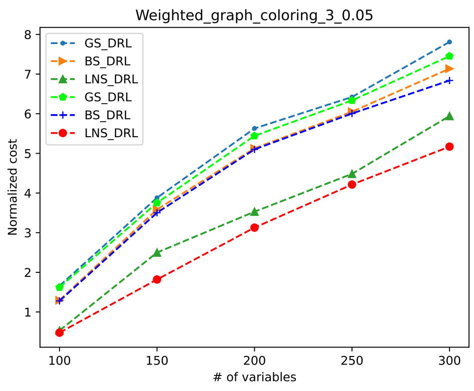


Fig. 16: Solution quality of the heuristics derived from our DRL model and the SL model on weighted graph coloring problems