Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 8003887, 10 pages
https://doi.org/10.1155/2022/8003887

Research Article

A GNN-Based Variable Partition Framework for DCOPs

Chun Chen,[1,2] **Li Ning**,[1] **Rong Zhou**,[3,4] **Yong Zhang**,[1] **Chan Zhou**,[1] **and Shengzhong Feng**[4]

[1]*Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, China*
[2]*University of Chinese Academy of Sciences, China*
[3]*Shenzhen Institute for Advanced Study, University of Electronic Science and Technology of China, China*
[4]*National Supercomputing Center in Shenzhen, China*

Correspondence should be addressed to Li Ning; li.ning@siat.ac.cn

Received 14 January 2022; Accepted 10 March 2022; Published 20 May 2022

Academic Editor: Yan Huo

Many problems of the Internet of Things (IoT), such as radio frequency allocation and sensor network, can be regarded as constraint optimal problems (COPs), which can be formulated as graphical representations. The scale of graph is large, which is hard to implement, and the information shared by all the variables is unsafe for all the variables running in an agent. On the other hand, supercomputers are playing a significant and growing role in various fields of large-scale processing tasks. When countering this scene, the supercomputers can accelerate to complete the task according to the distributed solution, where they divide the task into sub-tasks and each sub-task is running on an agent, such as a process or a computation node. However, finding an optimal distributed solution is difficult to minimize the completion time with the optimal computing resources. Putting the task on too many agents not only wastes resources but also increases the risk of attacks. Conversely, fewer agents may take too much time, which is unacceptable for users. Determining the number of agents needs to strike a balance between communication and computation. In this paper, we propose a new framework GVPNN for predicting the optimal numbers of agents for COPs and further provide the allocation from variable to agent. Experimental result shows the framework can learn the structure of the corresponding graphical representation well, and the 1-distant accuracy rate and the top 3 accuracy rate of GVPNN reach 74% and 70%, respectively.

1. Introduction

Constraint optimal problems (COPs) is to maximize its aggregated utility or minimize its cost. It has a wide range of applications, such as meeting scheduling, resource allocation, smart homes, sensor networks [1], and many IoT problems [2, 3], which has attracted the attention of many researchers. The distributed constraint optimization problems (DCOPs) [4, 5] is a distributed implementation for COP, which is a general model to govern the agent's autonomous behavior in a cooperative multi-agent system (MAS). COPs and DCOPs can be often represented graphically using one of the following representations: constraint graphs, factor graphs, or pseudo-trees [6]. In this paper, we call them as graphical representations. In all of these graphical representations, nodes in these graphs (i.e., variables and/or factors) are held by the agents which are participating in the optimization process.

For many applications in COPs, large-scale graphical representations are hard to implement with all the variables running in an agent, or even cannot be handled. In addition, it is unsafe to run all the variables in an agent sharing the information. Supercomputers can provide a good platform. To speed up the calculation on the supercomputer, an appropriate suggestion is important to provide for users with the number of computing resources (agents) and the allocation from variables to agents after learning the graphical representation. Therefore, users can adopt suggestions before submitting the task and purchase computing resources on supercomputers at reasonable prices. For the supercomputer, the resource can be efficiently utilized, and the tasks can be completed in the shortest possible time. How to find the optimal variable partition is very difficult because excessive agents are not only a waste resource but also increase the risk of being attacked. Otherwise, fewer agents may take too much time which is unacceptable for users.

In other words, the division of DCOPs is a plan that figures out the number of agents and the allocation from the variables to agents and have few work on the filed [7–10]. In the existing articles, many works focus on the DCOP algorithm and simply assigned one variable to an agent, which is negative for the large-scale DCOP. In ref [11], assume that the mapping of variables to agents is part of the model description, which means that variables that belong to each agent are given as an input. This assumption is reasonable in many applications where there are obvious and intuitive mappings. Take the smart home scheduling problem as an example; agents correspond to the different smart homes, and variables correspond to the different smart devices within each home. Under this scenario, it is reasonable for the agent to control all the variables, which are the devices in its home.

However, there are few scenarios like smart homes, and there may be more flexibility in other applications, which makes it hard to find the mapping from variables to agents. For example, imagine an application in which a group of unmanned robots needs to coordinate with each other to effectively survey an area. In this application, the agents correspond to robots, and the variables correspond to the different regions of the region to be solved. The domain for each variable may correspond to the different types of sensors to be used at different times to survey the region. Since a robot can survey any region, there are multiple possible assignments of regions to robots. That is, there are multiple possible mappings of variables to agents.

However, a good mapping is important as it has a significant impact on the completion time of an algorithm for the flexible scenarios. Choosing an optimal mapping may be prohibitively time-consuming as it is a NP-hard problem, as shown by Rust, Picard, and Ramparany [12]. To solve this problem, literature [13] proposed a time-efficient heuristic mapping algorithm (named MNA) based on the node's degree of the graphical representations. But this algorithm gives the agent number in advance and is only effective for messages passing algorithms such as Action-GDL, Max-Sum, or Bounded Max-Sum.

All the work discussed above focuses on the variable partition when the agent number is given. However, in many practical scenarios of large-scale distributed computing operations, different numbers of agents may lead to great differences in completion time, and the performance is not directly proportional to the number of agents. Therefore, it is very important to find the optimal number of agents for large-scale distributed computing operations, which can allocate computing resources of the supercomputers to users reasonably. In addition, compared with other DCOPs algorithms such as distributed random algorithm (DSA) [14] and distributed upper confidence tree algorithm (DUCT) [15], MNA has no advantage, and it is only effective when the distribution of the node degree in the graphical representations is casual. When the divergence of the node degree is unclear, the advantages of the algorithm cannot be highlighted.

In this paper, we commit to find the agent number when the DCOP algorithm is given and propose a new end-to-end variable partition framework. This framework employs graph neural networks (GNNs) to learn the structure of the corresponding graphical representation and then predicts the optimal number of agents and further provides the mapping from variables to agents.

The structure of the rest of this paper is as follows: Section 2 introduces the background and the definitions of the optimal number of agents, Section 3 describes the proposed framework and details the framework, Section 4 explains and analyzes the experimental results, and we conclude in Section 5.

2. Preliminaries

In this section, we introduce the background information of DCOPs and the objective of the paper.

2.1. Constraint Optimization Problem. A COP is a tuple $< X, D, F >$. X is a discrete and finite set of variables $\{x_1, x_2, \cdots, x_n\}$. D is a set of domains $\{D_1, D_2, \cdots, D_n\}$. Each domain D_i contains a discrete and finite set of values that can be assigned to variable x_i. We denote an assignment of value $d_{ij} \in D_i$ to x_i by an ordered pair $<x_i, d_{ij}>$. F is a set of relations (constraints). Each constraint $f_j \in \mathrm{F}$ defines a non-negative cost (or aggregated utility) for every possible value combination of a set of variables and is of the form $f_j : d_{i1} \times d_{i2} \times \cdots \times d_{ik} \longrightarrow F + \cup \{0\}$.

2.2. Distributed Constraint Optimization Problem. A DCOP is a distributed implementation for COPs, which is a tuple $<X, D, F, A, \mu>$. X, D, F is the same as COPs. A is a finite set of agents $\{a_1, a_2, \cdots, a_k\}$. $\mu : \mathrm{X} \longrightarrow \mathrm{A}$ is mapping from a set of nodes in the corresponding graphical representation to agents.

An optimal solution for COPs and DCOPs is an assignment with minimized cost or maximal utility. To be consistent with the literature, the aim for COPs and DCOPs in this paper default to minimize the cost, which is

$$X^* = \mathrm{argmin}_X \sum_{i=1}^{n} f_i(x_i), \tag{1}$$

2.3. Variable Partition. Given an agent number k, where $k > 1$, an arbitrary mapping manages a subgraph G_j of the graphical representations G, and k agents hold all the nodes which no nodes hold for different agents. The partition can be described as follows:

$$\begin{cases} U_{j=1}^{k} G_j = G \\ G_j \cap G_j' = \varnothing \quad \forall j' \neq j \end{cases}. \tag{2}$$

Given an arbitrary partition algorithm, the divergence of completion time may be huge for different number of agents.

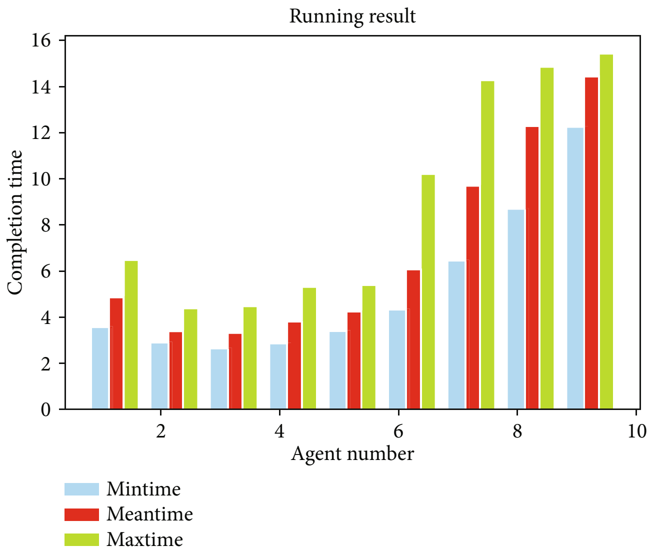

Figure 1: The completion time with different numbers of agents for a graph coloring problem, in which the graphical representation has 2 communities. For each partition solution, asynchronous execution DSA algorithm 10 rounds compute the mean time by formula (3), which is shown in the red bar. The min time (blue bar) and the max time (green bar) are minimal and maximal completion time in the 10 rounds.

Here, we give an example of a graph coloring problem (a special COP) on different agent numbers with partition by Girvan-Newman algorithm [16], in which an agent is asynchronous execution DSA algorithm on a process. The result can be shown in Figure 1. From the figure, we find that the optimal number is 3 (is computed by Formula (4)) and the mean completion time (is computed by Formula (3)) with 9 agents is more than 4 times slower than the mean completion time on the optimal number—3 agents.

2.4. The Optimal Agent Number. In this paper, the goal is to find the optimal number of agents and the mapping for an arbitrary COP. In this section, we first define the optimal number of agents and then verify the necessity of finding the optimal number of agents with an example.

To solve a COP with a large-scale graphical representation, an approximation algorithm can be the best choice. The completion time of the two experiments is different, and the variance is tiny from the expected value shown in the experimental result. Thus, we aim to find out the expected number of agents which are mostly performed the best. In this paper, the definition for the number of exceptional agents is given.

Given a COP problem p and the number of agent number k, it is defined that the valid completion time of each round i is the time t_{k_i} when the cost of the DCOP algorithm reaches 0 for the first time. In this paper, we supposed that each instance for DCOPs can be solved so that the cost can arrive at 0 and the total number for the round is n.

Since the Law of Large Numbers (LLN) describes the result of performing the same experiment many times, the average value of the results obtained from many experiments should be close to the expected value, and the average value will become closer as more experiments are performed in probability theory. In this paper, the expected completion time of agent number k is t_k, which is approximately defined as the average running time:

$$t_k = \frac{\sum_{i=1}^{n} t_{k_i}}{n}, \quad (3)$$

where t_{k_i} is the completion time time of the i round with k agent number and n the total number of rounds.

Then, we define the optimal number of agents with the minimum expected completion time as follows:

$$p_{ban} = \operatorname{argmin}(t_1, t_2, \cdots, t_k, \cdots, t_N), \quad (4)$$

where N is the total number of agents that can run on the supercomputer.

Finding the optimal agent number, the most immediate idea is searching for the number with a maximum module as most of the graphical representations for COPs are sparse and have the community structure. Girvan-Newman [16] proposes a module—Q value—which is an indicator to measure the quantity of clustering:

$$Q = \sum_i \left(e_{ii} + a_i^2\right), \quad (5)$$

where i is the i_th community, e_{ii} is the ratio of the edges of community i to all the edges of the original network, and a_i represents the ratio of all the edges connected with the vertices in the community i to the total number of edges.

From Formula (5), we find that the higher Q value, the better the corresponding community division results, and the best community division is the one with the largest Q value.

However, we find that the number of the optimal module may take more time than the agents of the other number. From Figure 2, we can find that there is no direct relationship between the minimum completion time and the

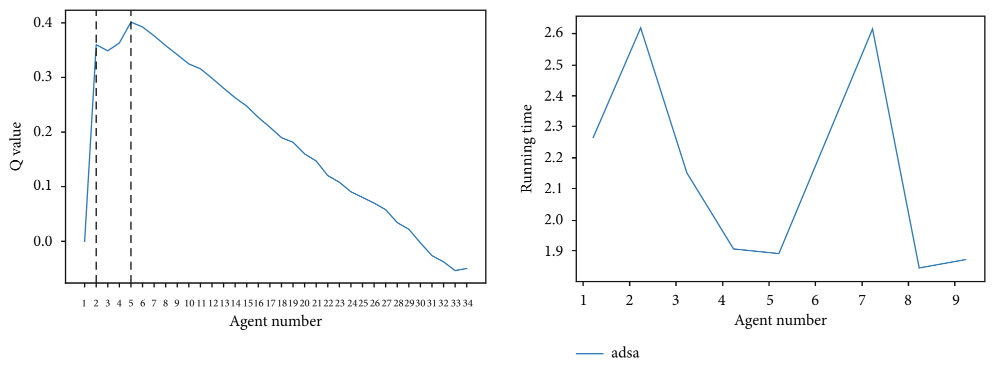

Figure 2: Q value and the mean completion time for a graph coloring problem with karate data.

module—Q value—in which the maximum q value is located on the agent number equal to 5, but the corresponding agent number of the minimum completion time is 8.

Therefore, this paper proposes a GVPNN framework to learn the characteristics of the graphical representations and find the best mapping for variables and agents, including the number of agents and specific mappings for each agent. In the next section, we will give more detail about the framework.

3. Graph Variable Partition Neural Network Framework

In this section, we first roughly introduce our Graph Variable Partition Neural Network Framework—GVPNN framework—which is based on graph neural networks [17–20]. When any COP arrives, this framework can abstract the COP to a graphical representation and then import the graphical representation into the graph neural networks to find the optimal number of agents, as shown in Figure 3. Finding the optimal number of agents is a graph classification problem in the domain of graph neural networks, which means that it predicts the label of an entire graph by learning a graph representation vector when given a group of graphs and their corrpending labels.

For an arbitrary node, it updates the representation vector by recursively aggregating and transforming feature vectors of its neighboring nodes in aggregate and combine stages. In GVPNN framework, we employ the GIN [21] and GraphSNN [22], which aggregate the feature of neighbor by multiset and aggregate the neighbor's feature and the overlap structure by the structural coefficients defined in Formula (8). After all the nodes are updated, the GVPNN obtained the feature of the whole graph. In the readout stage, all node features of the graph are converted into graph features, and then the optimal number of agents is predicted. Further, catch the optimal variable partition by Girvan-Newman algorithm.

3.1. Node Representation. The vector of a node representation is updated by recursively aggregating and transforming feature vectors of its neighboring nodes in the aggregate stage and combine stage. After $t+1$ aggregation iterations, we can capture the structure information of the neighborhood of the node's $(t+1)$-hop network. Formally, the node representation in the $(t+1)$_th layer can be represented as

$$\begin{aligned} a^{(t+1)} &= AGGREGATE^{(t+1)}\left(h^{(t)} : u \in N(v)\right), \\ h^{(t+1)} &= COMBINE^{(t+1)}\left(h^{(t)}, a^{(t+1)}\right). \end{aligned} \tag{6}$$

where $N(v)$ is a set of nodes adjacent to v.

In our paper, we employ two GNNs—GIN [21] and GraphSNN [22]—to update the node representation in our framework, as shown in Figure 4, which are all based on the Weisfeiler-Lehman test. The Weisfeiler-Lehman test is a test of graph isomorphism, which is an effective and computationally efficient test to verify whether two graphs are topologically identical in most cases. Node representation of u for the Weisfeiler-Lehman [23] test after $t+1$ iterate is the sub-tree structure of height $t+1$ rooted at the node u , which is updated by a hash function, which models injection multi-set functions of the neighbor aggregation.

Node representation for GIN is similar to 1- WL, in which the neighbor aggregation is also injected. Only when two nodes have the same sub-tree structure and have the same characteristics on the corresponding nodes, GIN will map these two nodes to the same location, where the sub-tree structures are recursively defined by the neighborhoods of the node. Representing a neighborhood as a multiset of feature vectors and treating the neighborhood aggregation as an aggregation function over multisets. To ensure injectivity, GIN sets the aggregate function to sum and the combination function as $(1+\epsilon^{(t+1)})$. The node representation is updated as

$$h_v^{(t+1)} = MLP^{(k)}\left(\left(1+\epsilon^{(k)}\right)h_v^{(t)} + \sum_{u \in \mathcal{N}(v)} h_u^{(t)}\right). \tag{7}$$

GraphSNN injects local structure into an aggregation scheme, considering not only the neighbor's feature but also the overlap subgraphs, which is more expressive than 1-WL. This GraphSNN define the structural *coefficients* ω (S_v, S_{uv}) for each vertex v and its local neighborhood, i.e., $\omega : S \times S^* \longrightarrow R$ such that $A_{vu} = \omega(S_v, S_{uv})$, which satisfies the properties of local closeness, local denseness,

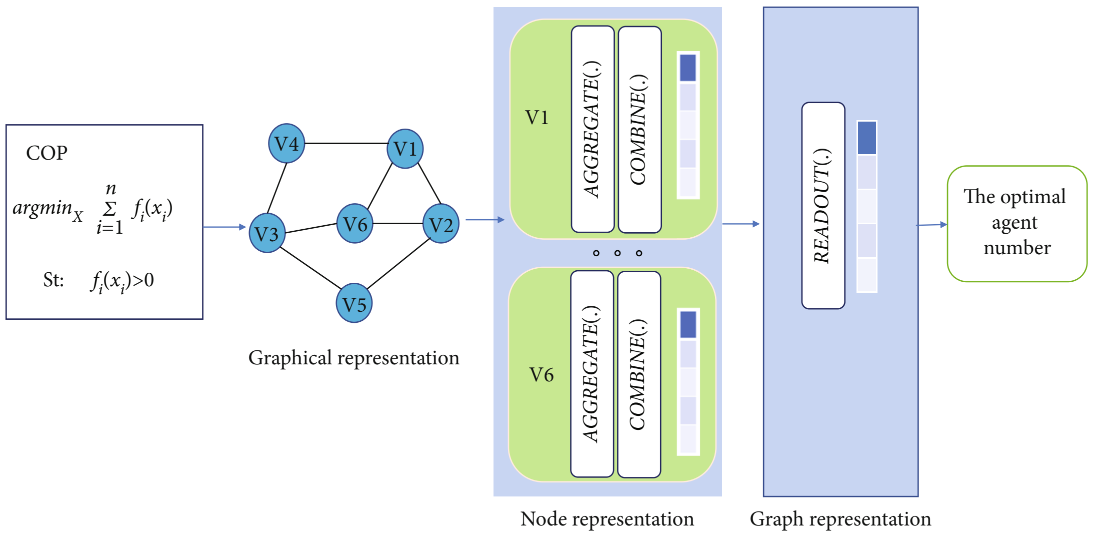

FIGURE 3: The framework for GVPNN.

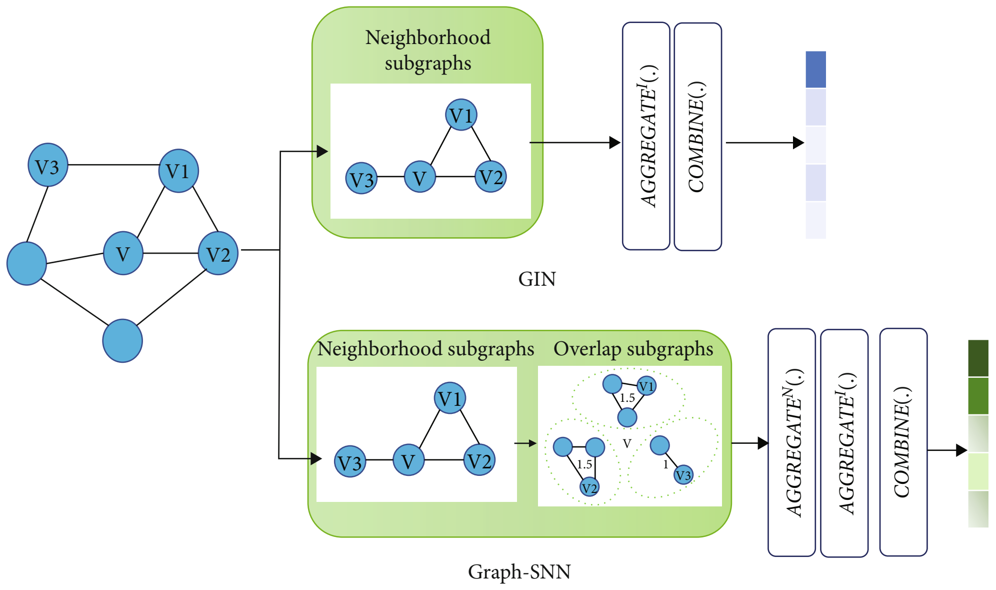

FIGURE 4: Node representation by GVPNN. For any node, GIN updates the representation vector by aggregating the neighbor's feature by multi-set, while GraphSNN updates the representation vector by aggregating the neighbor's feature and the overlap structure.

and isomorphic invariant:

$$\omega(S_v, S_{uv}) = \frac{|E_{vu}|}{|V_{vu}|\bullet|V_{vu}-1|}|V_{vu}|^{\lambda}, \tag{8}$$

where $\omega(S_v, S_{uv})$ is a structural coefficient for vertex v and its neighbors. S_v is the neighborhood subgraph for vertex v, and S_{uv} is the set of overlap subgraphs for vertex v and $\lambda > 0$.

GraphSNN also define a weighted adjacency matrix $\tilde{A} = (\tilde{A}_{vu})_{v,u\in V}$, where $\tilde{A}_{vu}$ is a normalized value of A_{vu} and $\tilde{A}_{vu} = A_{vu}/\sum_{u\in\mathcal{N}(v)}A_{vu}$. The node feature vector of v is updated by

$$\begin{aligned} m_a^{(t)} &= AGGREGATE^N\left(\left\{\left\{\left(\tilde{A}_{vu}, h_u^{(t)}\right)|u\in\mathcal{N}(v)\right\}\right\}\right) \\ m_v^{(t)} &= AGGREGATE^I\left(\left\{\left\{\tilde{A}_{vu}|u\in\mathcal{N}(v)\right\}\right\}\right) \\ h_v^{(t)}h_v^{(t+1)} &= COMBINE\left(m_v^{(t)}, m_a^{(t)}\right). \end{aligned} \tag{9}$$

$AGGREGATE^N(\cdot)$ and $AGGREGATE^I(\cdot)$ are two possibly different parameterized functions. Here, $m_a^{(t)}$ is a message aggregated from the neighbors of v and their structural coefficients, and $m_v^{(t)}$ is an "adjusted" message from v after performing an element-wise multiplication between $AGGREGATE^I(\cdot)$.

Specifically, GraphSNN details the $AGGREGATE^N(\cdot)$, $AGGREGATE^I(\cdot)$, and $COMBINE(\cdot)$. Thus, for each vertex v∈V, the feature vector at the $t+1\_th$ layer is generated by

$$h_v^{(t+1)} = MLP_\theta\left(\gamma^{(t)}\left(\sum_{u\in\mathcal{N}(v)}\tilde{A}_{vu}+1\right)h_v^{(t)} + \sum_{u\in\mathcal{N}(v)}\left(\tilde{A}_{vu}+1\right)h_u^{(t)}\right). \tag{10}$$

where $\gamma(t)$ is a learnable scalar parameter. Since $\mathcal{N}(v)$ refers to one-hop neighbors of v, one can stack multiple layers to handle more than the one-hop neighborhood. To ensure the injectivity in the feature aggregation,the graphSNN adds 1 into the first and aecond terms in the formula (10).

3.2. Graph Representation. For the graph classification problem, it is necessary to transform all the node features in the graph into graph features, and the representation of the entire graph is h_G:

$$h_G = READOUT\left(h^{(t+1)}\middle| v \in G\right), \tag{11}$$

where h_G is graph G representation vector and $READOUT$ represents a permutation invariant function and can also be a graph-level pooling function. This $READOUT$ function of the two GNNs is injective.

To consider all the structure information, the GVPNN framework utilizes the information from all iterations of the models and adopts an architecture like Jumping Knowledge Networks. The graph representations are concatenated across all the iterations/layers, and the $READOUT$ function is summing all node features from the same iterations:

$$h_G = CONCAT\left(SUM\left(\left\{h_v^{(t+1)} | v \in G\right\}\right)\right) | k = 0, 1, \cdots, K). \tag{12}$$

4. Experiment

All the experiments were executed on a server equipped by an Intel Xeon CPU 4110 with 20 cores of 2.20 GHz. The system is Linux 3.10.0, and all DCOP is implemented in the PyDCOP library.

4.1. Graph Coloring Benchmark. In the experiment, the graph coloring problem is used to benchmark coordination algorithms for our COP problem. In this work, in distributed graph coloring problems, variables in the graph must select their color (i.e., the state) from a set of possible colors (i.e., $x_i \in 1, \cdots, c$) and avoid conflicts (i.e., choosing the same color) with other variables. Thus, the cost is expressed as

$$U_m(x_m) = \gamma_m(x_m) - \sum_{i\in\mathcal{N}(m)\backslash m} x_i \otimes x_j \tag{13}$$

where,

$$x_i \otimes x_j = \begin{cases} 10, & \text{if} x_i = x_j \\ 0, & otherwise \end{cases}, \tag{14}$$

and $\gamma_m(x_m) < <1$ reflects the preference for any color in the absence of conflicts. As before, the goal is to find the state of each variable such that the total number of conflicts is minimized. In this work, we set $\gamma_m(x_m) = 0$ for the variable and set the conflict cost $x_i \otimes x_j$ =10 if two variable select the same coloring.

4.2. Dataset. In this paper, we choose three kinds of random graph datasets generated by networks to accomplish the graph coloring problem, and these graphs are undirected, unweighted, and connected:

(1) 991 instances of the 11-color random graph in the first dataset are generated by Stochastic Block model in which the number of communities is ranged from 2 to 6. The intra-block and cross-block probabilities are set to 0.001-0.002 and 0.1-0.3, respectively, and the size |v| of each graph was 200. Figure 5(a) give an example for a graph with 4 communities that the intra-block is 0.2 and cross-block probabilities is set to 0.001

(2) 454 instances of the 11-color random graph in the second dataset are generated by Erdős–Rényi model, which 307 with 200 nodes and 200-400 edges generated by gnm function and 147 instances of the 11-color random graph with excepted nodes of 200, and the probability between two nodes is 0.001 to 0.0025 generated by the gnp function. Figure 5(b) gives an example for a graph with 200 nodes and 350 edges

(3) 489 instances of the 11-color random graph with 200 nodes in the third dataset generated by the Barabasi Albert model. Figure 5(c) gives an example for a graph with 200 nodes

4.3. Labeling. For training in the framework GVPNN, we should first give a label for all the graphical representations. The labeling process includes the variable division, DCOP algorithm selection, and the optimal agent number calculation.

As the object is to find the optimal agent number with the minimum completion time, the initial allocation should be robust, and each part after being divided should not overlap. Many graphical representations are sparse and have a structure of communities. So in his work, the Girvan-Newman algorithm is adopt to divide each instance graph into k classes, k in [1,n], in which k= 1 means all the nodes in the same partition and each node is a apartition when k=N.

The Girvan-Newman algorithm [16] is proposed for graph partitioning in parallel computing, which minimizes the number of edges that run between processors. Giran-

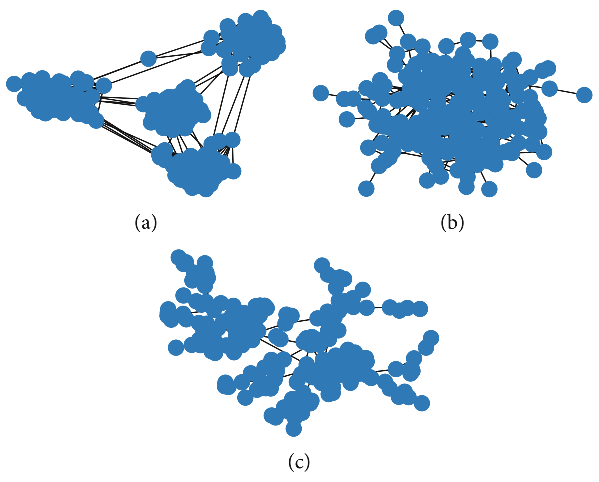

Figure 5: Random graphs generated by various models.

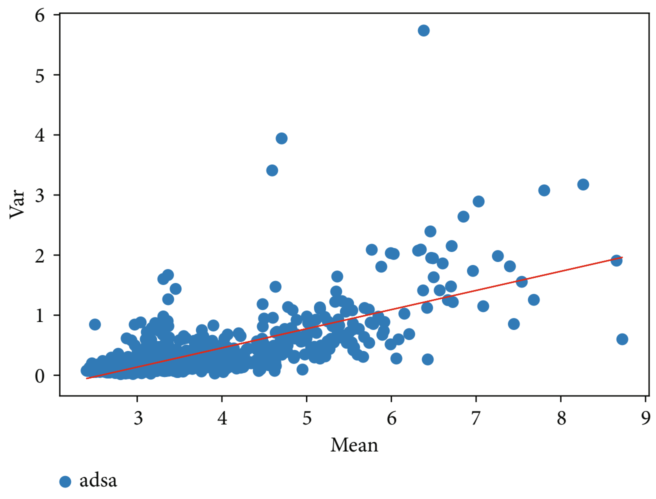

Figure 6: The relationship between the expected completion time and the variance.

Newman method is a hierarchical method, which detects communities by gradually removing edges from the original network, and the remaining connected networks are the communities, which ensure the non-overlap among communities. The algorithm deletes the edges that are most likely "between" communities, an edge as the number of shortest paths between pairs of nodes that run along with it, which is named "edge betweenness." After removing the edge with the highest score betweenness, the Girvan-Newman algorithm recalculates betweenness for all remaining edges, which are robust for the whole graph partition.

For algorithms, the field of classical DCOPs is mature, and lots of different solution algorithms have been proposed. According to whether the DCOP algorithm can guarantee the optimal solution, or whether it can trade optimality for shorter execution times, to generate the near-optimal solutions, the algorithms can be divided into complete algorithms and incomplete algorithms. For incomplete algorithms, there are three categories, such as search-based algorithms such as distributed random algorithm (DSA), maximum gain message (MGM), a reasoning-based algorithm such as max-sum algorithm, and sampling-based algorithm such as distributed upper confidence tree (DUCT) algorithm and distributed Gibbs (D-Gibbs) algorithm [24].

Because DSA is a good robust benchmark, and it tends to find high-quality solutions in practice, we choose to implement the DSA algorithm. Asynchronous actions may improve the performance of a DSA algorithm. So, this paper uses asynchronous DSA to get the completion time.

DSA requires an activation probability p before choosing new assignments, so we adopted $p = 0.7$ as reported in [14]. In addition, we used DSA version B because such a decision process is known to be more aggressive than other versions [14].

After picking the variable partition method and DCOP algorithm, we start to label the graphical representation of each instance of COPs. Specifically, we run the DSA algorithm for 10 rounds, in which each agent manages a partition on a process in the PyDCOP Library, and the optimal number of agents is calculated by the Formulas (3) and (4).

Because it is an approximate algorithm to run the DCOPs algorithm to find the best number for agents, we need to ensure the stability and effectiveness of the labeled result at first. To prove that the average time is stable and

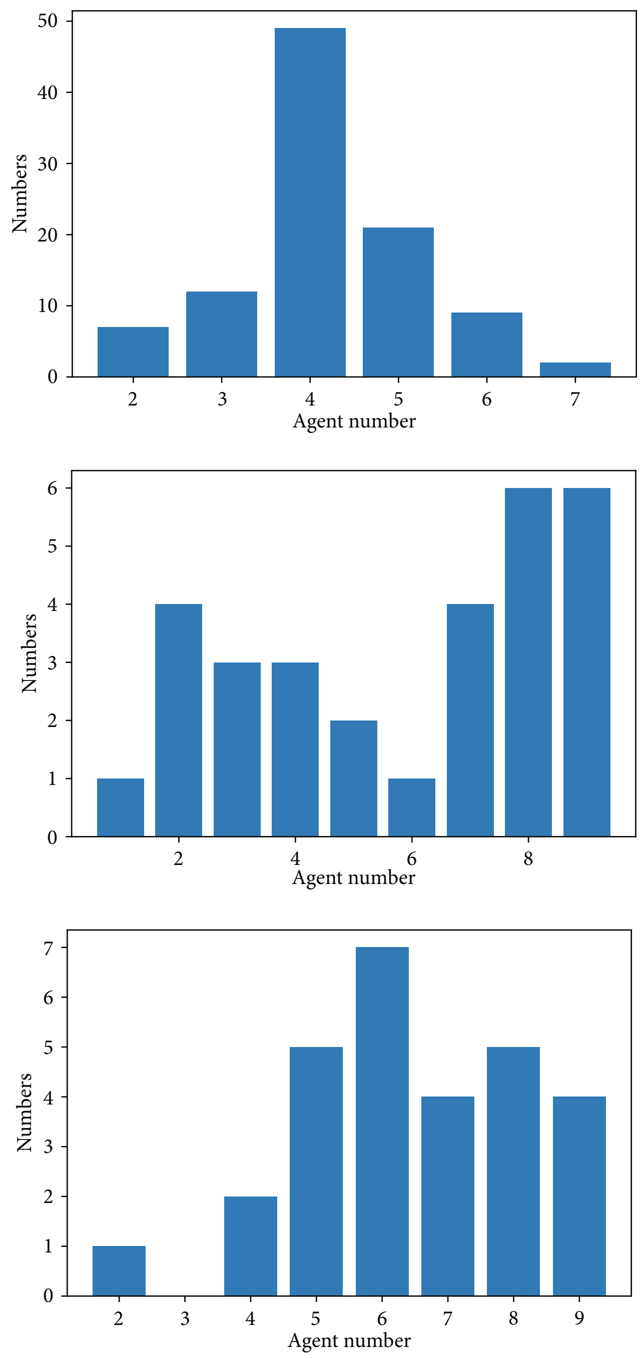

Figure 7: The best process distribution for random graphs.

effective, we analyze the relationship between the expected completion time of the distribution solution and the result variance, as shown in Figure 6.

From Figure 6, we found that the variance of the fitting curve coefficient was low, about 1/5 times the average value. The expected time can be considered the label of the graphical representations.

For the label distribution, we expect that it can be any value in the scope of supporting resources, not just the first three. Therefore, before marking the graph, we test various random graphs and make statistics on their distribution. Here, we give an example of optimal agent number distribution under different graph densities, as shown in Figure 7.

As shown in Figure 7, we find that the distribution of the optimal number of agents is diverse when analyzing graphs, which can meet the application of supercomputers. With the increase of edge, the number of agents may be limited to 3 agents.

5. Results

We train the GVPNN framework with GIN and GraphSNN on the graph coloring problem and compute the accuracies of the GNNs. As an agent is implemented in a process, the accuracy comes from existing documents:

$$Accuracy = \frac{NUM_{pred}}{NUM_{totle}}, \tag{15}$$

where NUM_{pred} is the $number\ pred_s = v_{opt}$.

What's more, we also use 1-distant accuracy and top 3 accuracy as the measurement standard. For the supercomputer, providing more or less than 1 process than the optimal process to COPs is also reasonable. So, 1-distant accuracy is set as follows:

$$Accuracy_{1dist} = \frac{NUM_{pred\_s}}{NUM_{totle}}, \tag{16}$$

where NUM_{pred_s} is the $number\ pred_s \in \{v_{opt} - 1, v_{opt}, v_{opt} + 1\}$.

Meanwhile, the completing time of the top 3 is always proximity. For the user, the prediction of the optimal agent number to a set with the top 3 for COPs works well. Thus, this paper shows the top 3 accuracy setting as follows:

$$Accuracy_{top3} = \frac{NUM_{pred\_top3}}{NUM_{totle}} \tag{17}$$

where NUM_{pred_top3} is the $number\ pred_s \in \{v_{top1}, v_{top2}, v_{top3}\}$.

Table 1 lists the results for GIN and GraphSNN. From the results, we found that the GVPNN can learn the structure of the graphical representations well. In terms of accuracy, GraphSNN network can reach $54.69\% \pm 6.59\%$ for the random graph with communities and improved 36% by GIN. For the random graph and the total graph dataset, the accuracy of GraphSNN and GIN is relatively close, which means that the overlap structure features in random graphs are not obvious.

For the supercomputer, the 1-distant precision GraphSNN can be improved more than GIN on three datasets, which is 48.5% for the random graph with communities, 65.1% for the random graph, and 31.2% for general graphs. For the user, the improvement of the accuracy of the top three GraphSNN is 23.4% for the random graph with communities, 12.5% for the random graph, and 4.9% for the total graph.

Table 1: Classification accuracies (%).

The random graph with communities			
Graphs	991		
Node degree avg/std/min/max	9.97/2.83/1/27		
Edge avg/std/min/max	997/29/896/1108		
	Accuracy	$Accuracy_{1dist}$	$Accuracy_{top3}$
GIN	40.2 ± 2.31	55.3 ± 5.72	69.3 ± 7.2
GraphSNN	54.69 ± 6.59	82.14 ± 6.84	85.57 ± 12.09
The random graph			
Graphs	943		
Node degree avg std/min/max	3.03/2.75/1/79		
Edge avg/std/min/max	303/109/199/728		
	Accuracy	$Accuracy_{1dist}$	$Accuracy_{top3}$
GIN	40.44 ± 1.83	45.2 ± 2.15	62.1 ± 3.60
GraphSNN	41.20 ± 1.66	74.65 ± 2.18	69.89 ± 4.74
The total graphs combine the random graph with communities and the random graph			
Graphs	1934		
Node degree avg/std/min/max	6.59/4.45/1/79		
Edge avg/std/min/max	658.86/355.7/199/1108		
	Accuracy	$Accuracy_{1dist}$	$Accuracy_{top3}$
GIN	39.6 ± 4.98	56.8 ± 3.96	67.3 ± 2.15
GraphSNN	41.11 ± 5.07	74.52 ± 3.75	70.63 ± 4.63

The result shows that GraphSNN can learn the structure of the graphical representations better than GIN and the predicted distribution of DCOPs' new graph is reasonable.

6. Conclusion

This paper presents an efficient framework for variable partition for DCOPs which is a pre-processing for DCOPs. COPs generates the labeled dataset by running the DCOPs algorithm based on the distribution during the Girvan-Newman algorithm. When a new COP arrives, GVPNN learns the graphical representation structure and further predicts the distribution. Experiments show that the framework can learn the architecture of graphical representations well, and the 1-distant accuracy of GraphSNN can reach 74.5%, and the top 3 accuracy can reach 70.6%. However, this framework only worked for Girvan-Neman community detection and DSA algorithm. In the following research, we will keep working to find the optimal variable partition with hybrid graph partition and DCOP algorithm.

Data Availability

The COP data (graphs) used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflict of interest.

Acknowledgments

This research was supported by the National Key Research and Development Project of China (Grant No. 2019YFB2102500) and the National Natural Science Foundation of China (Grant No. NSFC 12071460).

References

[1] W.-T. Chan, F. Y. L. Chin, D. Ye, Y. Zhang, and H. Zhu, "Greedy online frequency allocation in cellular networks," *Information Processing Letters*, vol. 102, no. 2-3, pp. 55–61, 2007.

[2] X. Zheng and Z. Cai, "Privacy-preserved data sharing towards multiple parties in industrial IoTs," *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 38, no. 5, pp. 968–979, 2020.

[3] J. Li, A. M. V. V. Sai, X. Cheng, W. Cheng, Z. Tian, and Y. Li, "Sampling-based approximate skyline query in sensor equipped IoT networks," *Tsinghua Science and Technology*, vol. 26, no. 2, pp. 219–229, 2021.

[4] F. Fioretto, E. Pontelli, and W. Yeoh, "Distributed constraint optimization problems and applications: a survey," *Journal of Artificial Intelligence Research*, vol. 61, pp. 623–698, 2018.

[5] Y. Zhang and Y. L. Francis, "A 1-local asymptotic 13/9-competitive algorithm for multi-coloring hexagonal graphs," *Algorithmica*, vol. 54, no. 4, pp. 557–567, 2009.

[6] A. Petcu and B. Faltings, "A distributed, complete method for multi-agent constraint optimization," in *CP 2004 - fifth international workshop on distributed constraint reasoning*, vol. 15, pp. 1–15, 2004.

[7] F. Fioretto, W. Yeoh, and E. Pontelli, "Multi-variable agent decomposition for DCOPs," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 30, pp. 2480–2486, Phoenix, Arizona, USA, 2016.

[8] P. Rust, G. Picard, and F. Ramparany, "On the deployment of factor graph elements to operate max-sum in dynamic ambient environments," in *International Conference on Autonomous Agents and Multiagent Systems*, pp. 116–137, Springer, Cham, 2017.

[9] P. Rust, G. Picard, and F. Ramparany, "Self-organized and resilient distribution of decisions over dynamic multi-agent systems," *International Workshop on Optimization in Multiagent Systems*, vol. 2018, 2018.

[10] A. Farinelli, A. Rogers, A. Petcu, and N. Jennings, "Decentralised coordination of low- power embedded devices using the max-sum algorithm," in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, vol. 2, pp. 639–646, Estoril, Portugal, 2008.

[11] F. Fioretto, W. Yeoh, and E. Pontelli, "A multiagent system approach to scheduling devices in smart homes," in *Proceedings of the 16th Conference on Autonomous Agents and Multi-Agent Systems*, pp. 981–989, Sao Paulo, Brazil, 2017.

[12] P. Rust, G. Picard, and F. Ramparany, "Using message-passing DCOP algorithms to solve energy-efficient smart environment configuration problems," in *In proceedings of the twenty-fifth inter- national joint conference on artificial intelligence, IJCAI16, 468474*, pp. 468–474, New York, USA, 2016.

[13] P. Agrawal, A. Kumar, and P. Varakantham, "Near-optimal decentralized power supply restoration in smart grids," in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, vol. 69, pp. 1275–1283, Istanbul, Turkey, 2015.

[14] W. Zhang, G. Wang, Z. Xing, and L. Wittenburg, "Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks," *Artificial Intelligence*, vol. 161, no. 1-2, pp. 55–87, 2005.

[15] B. Ottens, C. Dimitrakakis, and B. Faltings, "DUCT: an upper confidence bound approach to distributed constraint optimization problems," *ACM Transactions on Intelligent Systems and Technology*, vol. 8, no. 5, pp. 1–27, 2017.

[16] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review E*, vol. 69, no. 2, pp. 1–15, 2004.

[17] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *AAAI Conference on Artificial Intelligence*, pp. 4438–4445, New Orleans, LA, USA, 2018.

[18] R. Sato, "A survey on the expressive power of graph neural networks," 2020, http://arxiv.org/abs/2003.04078.

[19] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2021.

[20] B. M. Oloulade, J. Gao, J. Chen, T. Lyu, and R. Al-Sabri, "Graph neural architecture search: a survey," *Tsinghua Science and Technology*, vol. 27, no. 4, pp. 692–708, 2021.

[21] X. Keyulu, H. Weihua, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks," in *In International Conference on Learning Representations (ICLR)*, New Orleans, LA,USA, 2019.

[22] A. Wijesinghe and Q. Wang, "A new perspective on "how graph neural networks go beyond Weisfeiler-Lehman?"," in *In International Conference on Learning Representations (ICLR)*, 2021.

[23] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, no. 9, pp. 2539–2561, 2011.

[24] D. T. Nguyen, W. Yeoh, and H. C. Lau, "Distributed Gibbs: a memory-bounded sampling- based DCOP algorithm," in *In Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, vol. 69, pp. 167–174, Saint, MN, USA, 2013.