An End-to-End Multiple Hyper-parameters Prediction Method for Distributed Constraint Optimization Problem

Chun Chen[1(✉)], Yong Zhang[2], Li Ning[3], and Shengzhong Feng[4]

[1] Shenzhen Institute of Information Technology, Shenzhen, China
chun.chen@siat.ac.cn
[2] Shenzhen Institute of Advanced Technology, Shenzhen, China
[3] University of Electronic Science and Technology of China, Chengdu, China
[4] National Supercomputing Center in Shenzhen, Shenzhen, China

Abstract. Distributed Constraint Optimization Problem (DCOP) is an important model for multi-agents, has been widely used in various fields. When a large scale of DCOP implement on the supercomputer, various parameters need to choose, and the complement time vary widely for different combinations of parameters. Automatically provided accurate operating parameters for DCOP can improve the operation speed and enables the rational use of computational resources. However, the number of hyper-parameters of DCOP is huge, and correlation exists between hyper-parameters, thus make the prediction of multiply hyper-parameters difficult. In this paper we propose a new framework combine graph neural network and recurrent neural network. The performance shows that our framework can outperform the SODA method.

Keywords: multiply hyper-parameter · DCOP · Graph neural network · recurrent neural network

1 Introduction

The rapid development of artificial intelligence has attracted researchers' attention on multi-agent systems. The Distributed Constraint Optimization Problem (DCOP), as an important research direction on multi-agent, has been widely used in various fields in recent years. With the exponential growth of scale for DCOP, supercomputers have become the primary choice to cope with large-scale DCOP due to the storage and computing capacity of traditional computers. Nowadays, the common way to solve DCOP is to calculate the operating parameters by users who masters the domain knowledge and provide them to the supercomputing platform, which time-consuming and labor-intensive. So automatically provided accurate operating parameters for DCOPs can improve the operation speed and enables the rational use of computational resources.

The prediction of DCOP hyper-parameters is difficulty. First, DCOP involves many hyper-parameters, such as the DCOP algorithm and the corresponding parameters, the

J. S. Park et al. (Eds.): PDCAT 2023, LNEE 1112, pp. 202–214, 2024.
https://doi.org/10.1007/978-981-99-8211-0_19

graph partitioning algorithm. Second, correlation exists between hyper-parameters of DCOP. The performance under a single optimal parameter do not guarantee the overall optimal.

The multiply hyper-parameter prediction of DCOP can be simply considered as a multi-label recognition problem [3, 10]. However, the data of both image [4–9] and text problem are [11–14] regular, while the graph representation of DCOP is irregular data, so it is not possible to directly apply the present multi-label classification methods to the prediction for the hyper-parameter set of DCOP.

This paper addresses the difficulties of DCOP multiply hyper-parameter prediction and proposes a multiply hyper-parameter prediction framework combining graph neural network and recurrent neural network, whose contributions include the following:

(1) As there is no research on multi-parameter prediction for DCOPs, this paper gives the basic definition of the optimal parameter set and turns the DCOP multiply hyper-parameter prediction problem into a multi-label classification problem.

(2) For the multiply hyper-parameter prediction problem, this paper proposes a GRNN (Graph Recurrent Neural Networks) frameworks combining graph neural networks and recurrent neural networks, which considering the correlation of each parameter. The framework learned the features of the DCOP constraint graph by graph neural networks and handled the higher order parameter correlations by recurrent neural network.

(3) The extraction accuracy of graph feature vectors can affect the prediction accuracy of multiply hyper-parameter. This paper explores the influence on the number of layers of the graph neural network.

This paper is organized as follows: Sect. 2 introduces the basic theory of DCOP multiply hyper-parameter prediction and transforms the DCOP multiply hyper-parameter prediction problem into a multi-label classification problem, Sect. 3 introduces the multiply hyper-parameter prediction framework--- GRNN in detail, Sect. 4 analyzes the experimental results and discusses the experimental results and summarizes in Sect. 5.

2 Background

The performance of DCOP on supercomputing platforms are often associated with multiply hyper-parameter, such as graph partitioning algorithm [15], the DCOP algorithm, and the parameters corresponding to that algorithm. Before to predict the optimal hyper-parameter set, the definition of the optimal set of parameters OPT_{para} is given.

2.1 Definition of Optimal Set of Parameters

Given an DCOP instant and the overall sets of parameters for the instance $P_{para} = \{Para_1, Para_2, \ldots, Para_N\}$. For each set of parameters $Para_i$, which includes the execution method E_m, the algorithm A and the parameters corresponding to that algorithm $P_A = \{P_{A_1}, P_{A_2}, \ldots, P_{A_f}\}$, the graph partitioning algorithm G_p and the number of cores k, where $Para_i = [E_m, \left\{P_{A_1}, P_{A_2}, \ldots, P_{A_f}\right\}, G_p, k]$. The goal for this paper is to search the optimal set of parameters $Para_{opt}$ with the minimization completion time.

Firstly, we give a definition for the completion time of any instance under the set of parameters $Para_i$. If the $Para_i$ is selected, the instance implement the graph partitioning algorithm G_p to divide the DCOP's constrained graph into k parts and call the DCOP algorithm A and the parameters under the algorithm A to run the instance (synchronously or asynchronously) on k processes for a total of n rounds. Define the effective running time of the *i_th* round under the parameter $Para_i$ to be t_{pi_j}, which is the time for the cost function of DCOP to reach 0. This paper assumes that each instance of DCOP is solvable, i.e., there exists an effective time for the cost function to reach 0.

As the law of large numbers (LLN) in probability theory, where the average obtained from multiple experiments should be close to the expectation when performing the same experiment with multiple times, and the average will be closer to the expectation as the number of experiments increases. So, in this chapter, the completion time of any instance under the set of parameters is defined as the average completion time.

$$t_{pi} = \frac{\sum_{j=1}^{n} t_{pi_j}}{n} \tag{1}$$

where t_{pi} is the completion time of the *j_th* round of DCOP under the parameter set $Para_i$ and n is the total number of rounds run.

when the completion time of any instance under the set of parameters is defined then this paper defines the optimal set of parameters as follows:

$$Para_{opt} = argmin(t_{p1}, t_{p2}, \ldots, t_{pN}) \tag{2}$$

where N is the total capacity of the parameters P_{para}.

2.2 Comparison with Different Sets of Parameters

This paper introduces a small example, graph coloring problem, to compute $Para_{opt}$, and gives a representation of the completion time under different parameters sets. The example divides the constrained graph of DCOP into 1–4 subgraphs by the Giran-Newman algorithm or the METIS algorithm for graph partitioning. Each subgraph is then placed on the corresponding core to implement using the DCOP algorithm (DSA) either synchronously or asynchronously.

The result under some sets of parameters is showed in Fig. 1, which the example contains a total of 6 cases, and 10 rounds are executed for each set of parameters. The completion time is calculated by Eq. (2) and the optimal parameter for this example is obtained from Eq. (2) which is $P_{para} = \{\text{sy, Giran} - \text{Newman, DSA, p} = 0.7, 3\}$.

As shown in Fig. 1, the results under different sets of parameters are different and irregular, thus it hard to find the optimal parameters according to the traditional statistical methods. With the rapid development of neural networks, the multi-label classification solution problem has matured. In this paper, we will transform the multiply hyper-parameter prediction problem into a multi-label classification problem which using the optimal parameters as labels.

This paper gives the definition of multiply hyper-parameter prediction. For Each DCOP instance $G_i \in R_m$, which owns L subsets y in the parameter label space Y. The multi-parameter prediction task is to learn a function \$h: $h : R_m \rightarrow 2^{Y^D} = \{(x_i, y_i)|1 \leq$

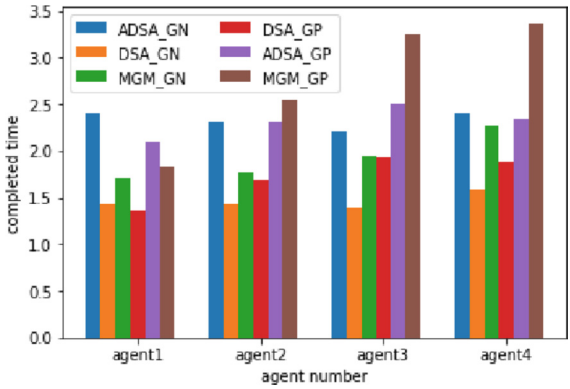

Fig. 1. The completion time under sets of parameters for graph coloring problem

$i \leq N\}$, where N is the total amount of training data, x_i is the vector in the input feature R_m of the i_th instance and $y_i \subset Y$ is a subset of the label space Y. Unlike the multi-classification problem where each instance is assigned only one label, the generalization of the multilabel problem provides multiple label assignments for each instance at the same time.

3 Multiply Hyper-parameter Prediction Model

In this section, a neural network framework---GRNN is proposed to predict the multiply hyper-parameters set, as shown in Fig. 2. The framework consists of three modules, the preprocessing module, the graph representations feature extraction module, and the multilabel prediction module. The preprocessing module converts the DCOP into a graph representation and extract the fixed-length feature vectors by the graph representations

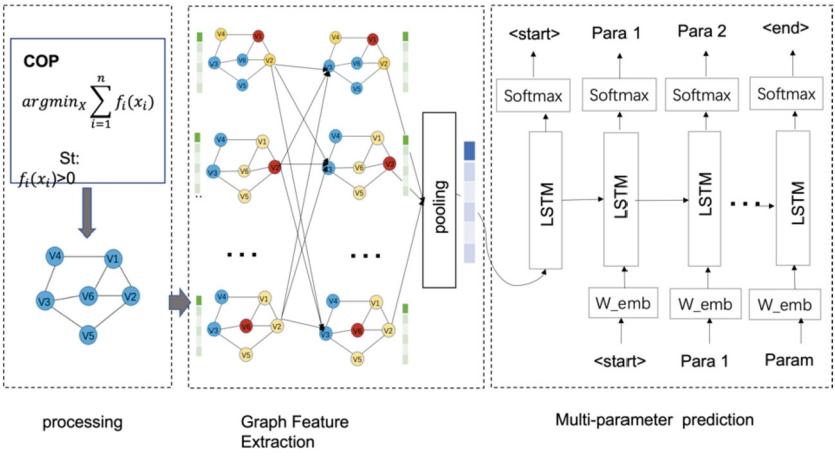

Fig. 2. Multiply hyper-parameter Prediction Framework Diagram

feature extraction module. Then, according to these feature vectors, the higher-order correlations between parameters are modeled in the multi-parameter prediction module.

3.1 Preprocessing Module

The DCOP cannot be solved directly in a graph neural network, [1, 2, 17] it needs to convert the DCOP into a graph representation. Since this paper may involve multiple graph representations, where different algorithms correspond to different graph representations. To ensure consistency, this paper uniformly converts the DCOP into a constraint graph.

3.2 Graph Feature Extraction Module

For the feature vector extraction of the DCOP, the GraphSNN is chosen in this paper. This network maps the local structure into the aggregation, considering not only the features of the neighbors but also the overlapping subgraphs. The feature extraction of DCOP includes node feature extraction as well as graph feature extraction.

3.2.1 Node Feature Extraction

Regarding node feature extraction, to better describe the neighborhood relationship between vertex v and its neighbors u, GraphSNN defines structural coefficients $\omega(S_v, S_{uv})$ for each vertex v, $\omega : S \times S^* \to R$.

$$\omega(S_v, S_{uv}) = \frac{|E_{vu}|}{(|V_{vu}||V_{vu} - 1|)|V_{vu}|^{\lambda}} \tag{3}$$

where $\omega(S_v, S_{uv})$ is the structure coefficient of vertex v and its neighbors. S_v is the neighborhood subgraph of vertex v and S_{uv} is the set of overlapping subgraphs of vertex v with $\lambda > 0$. $\omega(S_v, S_{uv})$ satisfies the properties of local compactness, local denseness, and isomorphism invariance. Let its adjacency matrix be $A = (A_{uv})_{uv \in V}$, where $A_{uv} = \omega(S_v, S_{uv})$.

GraphSNN also defines a weighted adjacency matrix $\mathrm{A} = (\overline{A_{uv}})_{uv \in V}$, where $\overline{A_{uv}}$ is the normalized value of A_{uv}, $\overline{A_{uv}} = \frac{A_{uv}}{\sum_{u \in N(v)} A_{uv}}$. So, the node eigenvectors of v are updated as

$$\begin{aligned} m_a^t &= \mathrm{Aggregate}^N(\{A_{vu}, h_u^t) | u \in N(v) \\ m_v^t &= \mathrm{Aggregate}^I(A_{vu} | u \in N(v)) h_v^t \\ h_{t+1}^v &= Combine(m_v^t, m_a^t) \end{aligned} \tag{4}$$

$\mathrm{Aggregate}^N(*)$ and $\mathrm{Aggregate}^I(*)$ are two different parameterized cumulative functions. Where m_a^t is the information aggregated from the neighbors v and their structural coefficients, m_v^t after performing the multiplication between the cumulative function

Aggregate$^I(*)$ and the multiplication between the eigenvectors, the "adjusted" message from v to account for the structural effects of its neighbors.

Specifically, the update function of GraphSNN for each vertex $v \in V$, whose node feature vector at $t+1$ layer is

$$h_v^{t+1} = MLP_\theta \gamma^t (\sum\nolimits_{v \in N(u)} A_{uv} + 1) h_v^t \sum\nolimits_{u \in N(v)} A_{uv} + 1) h_u^t) \tag{6}$$

where γ^t is a scalar parameter that can be learned. $N(v)$ refers to the one-hop neighbors v, and multiple layers can be stacked to handle more than one-hop neighbors. Note that to ensure the Monolicity of feature aggregation in the presence of structural coefficients, add 1 to the first and second terms of Eq. (6).

3.2.2 Graph Feature Extraction

For the graph classification problem, all node features in the graph need to be transformed into graph features, and the whole graph is represented as h_G.

$$h_G = Readout(h^k | v \in G) \tag{7}$$

where h_G is the graph G denotes the vector and Readout denotes the substitution invariant function, which can also be a graph-level pooling function.

The Readout function of the GraphSNN framework is single-shot. To consider all the structural information, the GraphSNN framework utilizes the information from all iterations of the model and uses an architecture similar to Jumping Knowledge Networks. The graphs represent connections in all iterations/layers and the *Readout* function sums all node features from the same iteration.

$$h_G = Concat(\sum\nolimits_{u \in N(v)} h_v^k | k = 0, 1, \ldots, K) \tag{8}$$

3.3 Hyper-parameter Prediction Module

After graphSNN obtains the representation graph vector of DCOP, the parameter prediction module uses the output graph vector of graphSNN as the initial state input for label prediction. Because there is some correlation before the parameters, for this reason LSTM is chosen in this paper to predict multiple parameters.

Despite the existence of several LSTM variants, this paper selects the standard LSTM, and applies an additional word embedding layer for the labels. The LSTM consists of three gates: an input gate i, an output gate o, and a forgetting gate f. The three gates work in concert to control what is read on the input, what is output, and what is forgotten, allowing some complex long-term relationships to be modeled.

$$\begin{aligned}
i &= \sigma(W^{(i)} x_t + U^{(i)} h_{t-1} + b^{(i)}) \\
o &= \sigma(W^{(o)} x_t + U^{(o)} h_{t-1} + b^{(o)}) \\
f &= \sigma(W^{(f)} x_t + U^{(f)} h_{t-1} + b^{(f)}) \\
u &= tanh(W^{(u)} x_t + U^{(u)} h_{t-1} + b^{(u)}) \\
c_t &= i \odot u + f \odot c_{t-1} \\
h_t &= o \odot \tanh(c_t)
\end{aligned} \tag{9}$$

where $\sigma(*)$ denotes element-by-element multiplication, which is a sigmoid function. $x_t \in R_d$ is the input of the lower layer at time step t. If the lower layer is a word embedding of parameters, then d can be the dimension of the labeled word vector or can be the hidden state dimension of the lower layer, if the lower layer is an LSTM. If there are q LSTM units, then for all types (i, o, f, u), $h_t \in R_q$, $W(*) \in R_q \times d$ and $b(*) \in R_q$.The memory cell c_t is the key in the LSTM, which maintains long-term dependencies while getting rid of the gradient disappearance/explosion problem. The forget gate f is used to erase some parts of the memory cell, while the input gate i and the output gate o control what is read from and written to the memory cell.

LSTM by linear transformation as Eq. (9) in each type (i, o, f, u) with additional terms$W(T)T$, where T is the output constraint graph feature from GNN with fixed dimension$t$, $W(T) \in R_q \times t$, q is the hidden dimension of LSTM, e.g. input The formula for the gate reads.

$$i = \sigma(W^{(i)}x_t + U^{(i)}h_{t-1} + W(T)T) \tag{10}$$

The label sequence prediction always starts with the tag < START >. At each time step, there is a SoftMax layer on top of the LSTM top layer. The probability of each label is calculated by first applying a linear transformation to the hidden state of the top LSTM layer.

Then, the tag with the highest probability is predicted. The prediction of the tag ends with the < END > tag. Therefore, for each input DCOP, a sequence of labels of different lengths is predicted. Ideally, the label sequence for each input DCOP matches exactly with the subset of labels belonging to that input DCOP.

4 Experimental Results and Analysis

4.1 Experimental Data

In the paper, we choose the graph coloring problem, a typical model of DCOP, to generate the corresponding dataset of this experiment. The datasets consist of two main parts, one part is the description of the DCOP and the corresponding constraint graph, and the other part is the label set which correspond to the multiply hyper-parameter. In this chapter, these two parts are introduced separately.

4.1.1 DCOP Problem Description

The graph coloring problem is a typical DCOP that has been widely used in coordination algorithms for sensor networks as well as benchmark, and many DCOP algorithms have also used it for performance comparisons.

In the distributed graph coloring problem, variables are located at the nodes of the constraint graph and choose a color (i.e., $x_i \in (1, ..., c)$ to avoid conflicts (i.e., choosing the same color) with other variables(nodes) connected to themselves through edges. Thus, the cost of each variable is expressed as

$$U_m(x_m) = \gamma_m(x_m) - \sum\nolimits_{i \in \frac{N(m)}{m}} x_i \otimes x_j \tag{11}$$

where, $x_i \otimes x_j = \begin{cases} 10 \ if \ x_i == x_j \\ 0 \quad otherwise \end{cases}$, $\gamma_m(x_m) \ll 1$, reflecting the preference of the variable for any color in the absence of conflict. Consistent with the DCOP definition, the goal is to find the state of each variable that minimizes the conflict. In this experiment, this paper sets $\gamma_m(x_m) = 0$ and sets the edge conflict cost, i.e., two nodes with edges in the constraint graph choose the same color, $x_i \otimes x_j = 10$.

4.1.2 Random Graph Generation Based on Graph Coloring DCOP

In this experiment, three kinds of undirected, unweighted and connected random graphs are generated by network further four datasets are selected which cover the basic random graphs, etc.

1) dataset contains 438 random graph instances of 11 colors generated by the Erdős - Rényi model, which 316 instances are generated by the gnm function with 200 nodes and 200–400 edges, and 122 instances are generated by the gnp function with an link probability from 0.1–0.2.
2) The second dataset has 29 instances of 11-color random graph coloring, which consists of instances generated by the Small wolrd model.
3) The third dataset has 100 instances of 11-color random graph coloring. The instances are generated by the Barabasi Albert model. The random graph degree generated by this model has a power-law distribution.
4) The fourth dataset are assembled the above three datasets.

4.1.3 Hyper-parameter Set Generation and Validity

Since the goal of this paper is to find the set of optimal hyper-parameters, and the framework is a supervised learning framework, this section starts by labeling each random graph with the original label. To ensure the accuracy of the prediction, this chapter needs to ensure the validity of the labels and that the initial assignment is robust. To verify the validity of the framework, this paper selected DCOP algorithms such as DSA, MGM, etc., and the Giran- Newman algorithm as well as the METIS graph partitioning algorithm.

To ensure the validity of the labels, this paper conducts 10 trials for each set of hyper-parameters and hopes that the results of each set of hyper-parameters on experiments are stable, i.e., the variance is not large. In this paper, we analyze the results of each set of hyper-parameters as shown in Fig. 3.

From Fig. 3, we finds that the variance of the fitted curve coefficients is low, about 0.26 times the mean. The expected time can be considered as the label of the constrained graph.

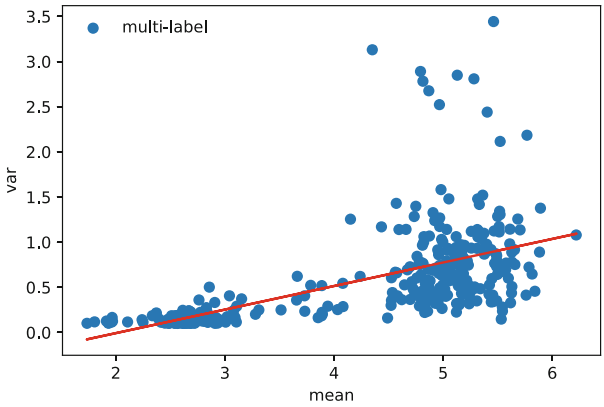

Fig. 3. The relationship between expected running time and variance

For the label distribution, we run multiple DCOP problems in this paper and find a more uniform parameter distribution, as shown in Fig. 4. The figure shows the optimal parameter distribution of the dataset DCOP_{BA} after multiple rounds of experiments.

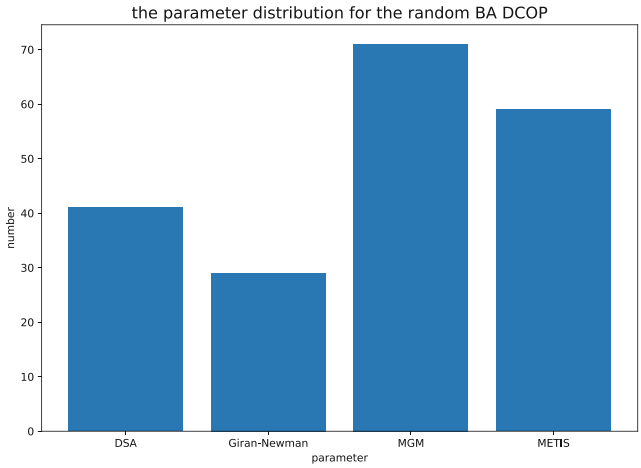

Fig. 4. The distribution for BA parameter

4.1.4 Dataset Description

For this purpose, the structural information of the four datasets trained and the labeling information are described in this paper as follows, as shown in the Table 1, Where DCOP_{ER} is dataset 1, a random graph generated for the Erdős - Rényi model, DCOP_{BA} is dataset 2, a random graph generated by the Barabasi Albert model, DCOP_{SW} is dataset 3, a random graph instance generated for the SW model, DCOP_{ALL} is data set 4, which is the merge of the above four data sets.

Table 1. Dataset description

data	number	Degree of nodes	Number of edges	Number of hyper-parameters
$DCOP_{ER}$	438	3.6/1.94/1/15	359.88/81.96/200/546	3
$DCOP_{BA}$	100	1.99/2.81/1/51	199/0/199/199/199	4
$DCOP_{SW}$	29	5.56/1.44/4/12	556.28/1.44/4/12	3
$DCOP_{ALL}$	567	3.42/2.24/1/51	341.55/110.07/199/737	4

4.2 Experimental Results and Analysis

4.2.1 Evaluation Index

To fairly compare the results of other methods, the average precision (CP) is reported in this section for performance evaluation.

$$CP = \frac{1}{c}\sum_i \frac{N_i^c}{N_i^p} \tag{12}$$

4.2.2 Parameter Setting and Running Platform

All experiments were performed on a server with an Intel Xeon CPU 4110 equipped with 20 2.20 GHz cores. The system was Linux 3.10.0 and all DCOPs were implemented in the PyDCOP library. All multiclassification graph neural networks were implemented in pytorch.

This paper uses the Adam optimizer [16] with $\lambda = 1$. For all datasets of DCOP, the model was trained for 500 periods with a learning rate of 0.01, a loss rate of 0.5, a hidden layer of 256, and $\gamma = 0.1$. This chapter select the random division method, i.e., the graph is randomly divided into 60\%, 20\% and 20\% for training, validation, and testing.

4.2.3 Analysis of Experimental Results

Since this paper is required to calculate the optimal parameters, in order to verify the effectiveness of the algorithm, two common methods are compared: 1) ordinary dichotomous GNN, i.e., GNN is used to generate the graph features of the DCOP constraint graph, for each label, which is treated as a one-by-one dichotomous classification in this chapter. (2) Since this paper does not involve many parameters, the multi-classification method graphSNN is chosen as a comparison experiment. Since the parameters involved in this chapter are less, the method converts the multi-parameter prediction into a multi-classification method and puts it into the graphSNN network. The results are shown in Table 2.

Table 2 lists the accuracy results of multiply hyper-parameters prediction for different datasets. The results show that the accuracy of both the multiclassification algorithm---GraphSNN and the GRNN algorithm on all the datasets is higher than that of the ordinary binary classification algorithm GNN. For the GraphSNN algorithm and GRNN

Table 2. The accuracy for the prediction

dataset	Algorithm	Accuracy
$DCOP_{ER}$	GNN	76.53 ± 3.12
	GraphSNN	93.84 ± 2.28
	GRNN	84.74 ± 4.34
$DCOP_{BA}$	GNN	42.31 ± 4.58
	GraphSNN	46.0 ± 11.13
	GRNN	58.67 ± 2.66
$DCOP_{SW}$	GNN	70.75 ± 2.94
	GraphSNN	91.16 ± 5.49
	GRNN	86.4 ± 10.88
$DCOP_{ALL}$	GNN	73.17 ± 1.38
	GraphSNN	81.13 ± 2.44
	GRNN	93.52 ± 2.47

algorithm, the accuracy of the recurrent neural network does not play a larger role when there are few labels, and its accuracy is not as good as that of the GraphSNN, and its ER dataset and WS dataset both perform less well than the GraphSNN when there are only three labels. In the ER dataset, the accuracy of GraphSNN is 10.7% higher than that of GRNN method, and in the SW dataset, the accuracy is 5.8% higher. However, the accuracy of GRNN improves as the number of labels increases, and it improves by 27.54% in the BA dataset and 14% in the total dataset compared to the GraphSNN.

4.2.4 The Effect of Graph Neural Network Depth on Performance

The exaction of graph features is one of the most important factors affecting multi-parameter prediction, and different graph neural network embedding operations have an impact on the performance of experimental results. Specifically, different layers of graph neural networks have different obtained graph features. For this reason, this section explores the effect of different neural network layers on the prediction results, as shown in the Table 3.

Table 3. The accuracy on different neural network layers

Dataset	2 layers	3 layers	4 layers
$DCOP_{ER}$	84.74 ± 4.34	82.36 ± 3.74	81.47 ± 7.41
$DCOP_{BA}$	58.67 ± 2.66	57.49 ± 4.51	55.16 ± 2.73
$DCOP_{SW}$	86.4 ± 10.88	83.14 ± 7.29	82.46 ± 5.29
$DCOP_{ALL}$	93.52 ± 2.47	91.45 ± 4.28	89.54 ± 6.85

It finds that the accuracy of the prediction results to decrease to some extent when increasing the number of layers of the convolutional layers. The possible reasons for this are mainly due to the following two points. One is that the number of parameters will also become dramatically larger due to the increase in the number of layers of the convolution of the graph which will cause the overfitting phenomenon to some extent. Second, although the method in this paper greatly alleviates the over-smoothing problem, but it does not avoid the over-smoothing problem, and the deepening of the convolutional layer will add the over-smoothing problem leads to performance degradation.

5 Summary

Multiply hyper-parameter prediction of DCOP is an important subject, and its high accuracy can be an effective guarantee of DCOP. This paper first demonstrates experimentally that DCOP has large differences in its operation results under multiple sets of parameters and that traditional methods cannot effectively predict multiple parameters accurately. Then transforms the multiply hyper-parameter prediction problem of DCOP into a multi-label prediction problem and proposes a novel neural network-based multi-label classification method. Experiments demonstrate the effectiveness of the methods from both qualitative and quantitative perspectives, respectively.

However, the GRNN is a supervised prediction models, which require multiple runs of DCOP to generate the corresponding training data, and the data acquisition cost is relatively expensive. In the future, we will learn new techniques such as semi-supervised graphical neural networks to solve the problem.

References

1. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search (2019). arXiv:1806.09055
2. Schweidtmann, A.M., Rittig, J.G., König, A., et al.: Graph neural networks for prediction of fuel ignition quality. Energy Fuels **34**, 11395–11407 (2020)
3. Zhang, J., Wu, Q., Shen, C., et al.: Multilabel image classification with regional latent semantic dependencies. IEEE Trans. Multimed. **20**, 2801–2813 (2018)
4. Chen, Z.M., Wei, X.S., Wang, P., et al.: Multi-label image recognition with graph convolutional networks. IEEE/CVF Conf. Comput. Vision Pattern Recogn. (CVPR) **2019**, 5172–5181 (2019)
5. Wang, Y., Xie, Y., Liu, Y., et al.: Fast graph convolution network based multi-label image recognition via cross-modal fusion. In: Proceedings of the 29th ACM International Conference on Information & Knowledge Management (2020)
6. Chen, T., Xu, M., Hui, X., et al.: Learning semantic-specific graph representation for multi-label image recognition. In: 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pp. 522–531 (2019)
7. You, R., Guo, Z., Cui, L., et al.: Cross-modality attention with semantic graph embedding for multi-label classification. arXiv:abs/1912.07872 (2020)
8. Zhang, M., Shao, H.C., Song, G., et al.: Top-1 solution of multi-moments in time challenge. arXiv:2003.05837 (2019)
9. Zhao, J., Yan, K., Zhao, Y., et al.: Transformer-based dual relation graph for multi-label image recognition. IEEE/CVF Int. Conf. Comput. Vision (ICCV) **2021**, 163–172 (2021)

10. Kim, Y.: Convolutional neural networks for sentence classification. In: EMNLP (2014)
11. Lai, S., Xu, L., Liu, K., et al.: Recurrent convolutional neural networks for text classification. In: AAAI (2015)
12. Chen, G., Ye, D., Xing, Z., et al.: Ensemble application of convolutional and recurrent neural networks for multi-label text categorization. Int. J. Conf. Neural Netw. (IJCNN) **2017**, 2377–2383 (2017)
13. Yang, Z., Yang, D., Dyer, C., et al.: Hierarchical attention networks for document classification. In: NAACL (2016)
14. Sun, C., Qiu, X., Xu, Y., Huang, X.: How to fine-tune BERT for text classification? In: Sun, M., Huang, X., Ji, H., Liu, Z., Liu, Y. (eds.) Chinese Computational Linguistics: 18th China National Conference, CCL 2019, Kunming, China, October 18–20, 2019, Proceedings, pp. 194–206. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-32381-3_16
15. Pizzuti, C.: Evolutionary computation for community detection in networks: A review. IEEE Trans. Evol. Comput. **22**(3), 464–483 (2018). https://doi.org/10.1109/TEVC.2017.2737600
16. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR abs/1412.6980 (2015)
17. Pei, H., Wei, B., Chang, K.C.C., et al.: Geom-GCN: geometric graph convolutional networks. arXiv:2002.05287 (2020)