Dual-Mechanism Balanced Graph Partitioning: Contrastive Learning for Coarse Partition Generation and Reinforcement Learning for Fine-Tuning

Yongqing Wang, Guosun Zeng*
School of Computer Science and Technology, Tongji University, Shanghai, China
{2432007, gszeng}@tongji.edu.cn

Abstract— **Traditional balanced graph partitioning methods are mainly heuristic and experience-driven. Recent advances adopt deep learning, but existing approaches suffer from loss function mismatch and modeling redundancy. To address these issues, this paper proposes a Dual-Mechanism Balanced Graph Partitioning method (DMBGP), combining contrastive learning for coarse partition generation and reinforcement learning(RL) for fine-tuning. Specifically, positive and negative samples are leveraged to train a graph neural network via contrastive learning, with a tailored partition contrastive loss to extract label-agnostic features and generate initial partitions. RL then performs pairwise fine-tuning of adjacent partitions, and the final global result is obtained through iterative optimization over all adjacent pairs. The pairwise tuning paradigm alleviates modeling redundancy, decouples the state-action space from partition cardinality, and supports arbitrary k-partitioning tasks. Extensive experiments on diverse datasets demonstrate that DMBGP outperforms traditional heuristic methods and state-of-the-art deep learning methods, providing an efficient solution for deep learning-enabled balanced graph partitioning.**

Keywords—**Contrastive Learning, Reinforcement Learning, Balanced Graph Partitioning**

I. INTRODUCTION

Graph partitioning refers to dividing a graph into disjoint partitions. Balanced partitioning requires that each partition contains an equal number of vertices while minimizing the number of edges between partitions. As a fundamental problem in network analysis, graph partitioning finds practical applications in numerous fields, such as system design of online social networks, real-time control of transportation networks, and reasoning computation of knowledge graphs.

Balanced graph partitioning is an NP problem, and researchers mainly resort to heuristic methods to obtain near-optimal solutions. Traditional heuristic methods include hashing methods, bipartitioning methods, intelligent heuristic methods, multilevel partitioning methods, and label propagation methods. Following fixed strategies designed by experts, these methods generate high-quality partitioning results in a short time. However, such methods inherently have significant limitations: first, the partitioning strategy is single and fixed, lacking adaptive capabilities for diverse graph structures. Second, the partitioning objectives are tightly coupled with the strategies, making it impossible to adjust the objectives freely. Finally, these methods are highly dependent on expert experience. Researchers are required to conduct long-term observations and repeated experiments on graph data in specific domains to summarize effective adaptive strategies.

The rise of deep learning provides a new approach to address these bottlenecks, among which the combination of Graph Neural Networks (GNNs) and Reinforcement Learning (RL) is particularly promising. GNNs adapt to irregular graph topologies through message-passing and graph convolution. Nodes update their features by interacting with neighbors, capturing node dependencies and global structures to boost adaptability. RL enables an agent to continuously optimize strategies through environmental interaction, eliminating manual expert design. By adjusting the reward function, partitioning objectives can be adjusted freely. Essentially, graph partitioning can be framed as a vertex classification task, making GNN-RL combinations naturally suitable for this task.

Despite the fruitful achievements of deep learning in graph computing, its application to balanced graph partitioning faces two core bottlenecks: First, loss function mismatch: Graph partitioning requires partition labels to be permutation-invariant—swapping any two partition labels does not change the essential result. However, existing methods use label-based supervised losses (e.g., squared difference between predicted and ground-truth labels), leading the model to learn different weights when partition labels are swapped, failing to capture label-agnostic partitioning features. Second, modeling redundancy: Mainstream RL methods adopt global state spaces that include all partition information, resulting in excessive irrelevant data interference when evaluating vertex migration gains. Additionally, the state-action space size is coupled with the number of partitions k, reducing decision efficiency and limiting adaptability to arbitrary k-partitioning tasks.

To address the aforementioned research gaps, this paper proposes DMBGP. Our key contributions are:

- **Partition Contrastive Learning with Permutation Invariance**: A customized contrastive loss maximizes intra-partition feature similarity and minimizes inter-partition similarity, eliminating partition label interference to satisfy the task's permutation invariance requirement and enable learning of general, label-agnostic features.
- **RL-based Fine-tuning Focused on Partition Pairs**: Focusing on adjacent partition pairs, the MDP state space is limited to boundary vertices and associated partitions, and the action space is simplified to binary decisions (retain/migrate). This reduces redundancy, decouples the state-action space from k and improves decision efficiency.
- **Excellent Performance in Balancing and Connectivity**: Experiments on multiple graph datasets verify DMBGP outperforms state-of-the-art methods in cut ratio, modularity, and other core metrics, with strong generalization to arbitrary k values.

II. RELATED WORK

This section reviews relevant work from two perspectives: traditional heuristic methods and modern deep learning-based methods, and further analyzes the core bottlenecks in current research.

A. Heuristic Rule-based Graph Partitioning Methods

Heuristic methods generate near-optimal solutions for specific graph structures by expert-summarized rules, but lack theoretical quality guarantees and adaptability to diverse

graphs. Classic categories include: Random Hashing Partitioning Methods [1,2,3,4]: These methods randomly assign graph vertices to different partitions, but with low quality. Bi-partitioning Methods [5,6,7]: These methods first bisect the entire graph into two subgraphs, then recursively bisect each subgraph until the graph is divided into k equal-sized partitions. Representative algorithms include KL [8], FM [9], and their variants. Intelligent Heuristic Methods [10,11]: Inspired by biological intelligence in nature, these methods address the balanced graph partitioning problem by simulating natural evolutionary or search processes. Typical examples include tabu search, simulated annealing, evolutionary algorithms, and neighbor expansion (NE) [12,13,14]. Multilevel Partitioning Methods: The execution of these methods consists of three core steps: coarsening, partitioning, and refinement [15,16]. The Metis [17] multilevel partitioning method is the gold standard for graph partitioning, with most improved methods benchmarked against it. KaHIP [18] and Scotch [19] have also achieved excellent performance through optimization of the multilevel framework. As the most accurate and high-quality algorithms among traditional methods, multilevel graph partitioning methods exhibit strong robustness.

B. Deep Learning-based Graph Partitioning Methods

Machine learning in the AI era offers a novel approach to graph partitioning, freeing researchers from manual heuristic rules. By autonomously learning graph structural features and partitioning logic, these methods significantly improve partitioning adaptability and quality, mainly classified into supervised and unsupervised learning. RL, a key unsupervised subfield, is highly valuable for partitioning optimization.

Supervised methods rely on fully labeled data, learning explicit patterns for partitioning and excelling at capturing task-specific features. Fan et al. [20] modeled the partitioning objective as a multivariate function, training its "black-box" expression via polynomial regression on graph task operation logs. Qin et al. [21] proposed PR-GPT, a pre-train-fine-tune framework. It learns general community features from small labeled graphs offline, generalizes to large graphs for initial partitions, and refines results with efficient strategies.

Unsupervised methods mine inherent graph structural correlations or clustering properties, suitable for label-scarce scenarios. Nazi et al. [22] proposed GAP, targeting generalization to unseen graphs. It designs a differentiable loss integrating normalized cut relaxation and balance constraints, and optimizes network parameters via backpropagation, achieving fast inference while maintaining competitive partitioning quality. Bianchi et al. [23] proposed MinCutPool, addressing high complexity of spectral clustering's Laplacian eigen-decomposition by transforming normalized min-cut into a differentiable objective for hierarchical pooling considering topology and node features. Tsitsulin et al. [24] proposed DMoN, converting modularity maximization into a differentiable function for end-to-end unsupervised clustering, with collapse regularization to avoid degeneration.

Within the unsupervised learning framework, RL leverages agent-environment interaction to model partitioning as sequential decision-making, with tailored reward functions for complex scenarios. Dai et al. [25] proposed a general RL framework for graph combinatorial optimization, using graph embedding for global features and learning greedy incremental solution strategies. For balanced partitioning, Zhou et al. [26] proposed RLCut with multi-agent collaboration, coordinating local accuracy and global balance via global state perception for distributed large graphs. Shah et al. [27] proposed NeuroCUT, modeling vertex partition adjustment as MDP based on initial clustering to learn refined local strategies. Gatti et al. [28] adopted hierarchical recursive bisection, using RL to optimize subgraph balance and edge cut at each level.

III. PRELIMINARY

A. Basic Concepts and Assumptions of Graphs

This paper focuses on attribute-less graphs $G(V,E)$, where V is the vertex set and E is the edge set. v_i denotes vertex i, and e_{ij} denotes the edge connecting v_i and v_j. $N(v_i)$ represents the set of neighbor vertices of v_i. $\deg(v_i)$ denotes the degree of vertex v_i, with $\deg(v_i) = |N(v_i)|$. Graph partitioning refers to dividing graph G into k disjoint subsets. Let P denote the set of partitions obtained, where each partition is represented by P_i. Then $P = \{P_1, P_2, \dots, P_k\}(\bigcup_{i=1}^{k} P_i = G, P_i \cap P_j = \emptyset \; for \; i \neq j)$. The vertex set and edge set of partition P_i are denoted as V_i and E_i respectively. $cut(P_s, P_t)$ represents the set of cut edges between two partitions, defined as $cut(P_s, P_t) = \{e_{ij} | v_i \in V_s, v_j \in V_t\}$. This paper adopts edge-cut partitioning with no vertex or edge duplication allowed.

B. Source of Graph Partitioning Labeled Data

Labeled training data is generated by partitioning graph datasets from SNAP and Network Repository using classic algorithms. Specifically, for graph $G(V,E)$, classic algorithms are applied to generate the partitioned graph $G(V,E,Y)$, where $y_i \in Y$ denotes the partition label of v_i.

C. Feature Characterization of Graph Vertex Attributes

For attribute-less graphs, we construct initial vertex features by concatenating statistical features and structural features.

Statistical features are derived from local and global graph structure statistics, including three core indicators: Degree Centrality: $\deg(v_i)$, the number of direct neighbors of v_i. Clustering Coefficient: $\mathrm{CC}(v_i) = \frac{2T(v_i)}{\deg(v_i)(deg(v_i)-1)}$, where $T(v_i)$ represents the number of edges between the neighbors of v_i. PageRank Score: The iterative calculation formula is $\mathrm{PR}(v_i) = \frac{1-d}{|V|} + r\sum_{v_j \in N(v_i)} \frac{PR(v_j)}{\deg(v_j)}$, where d is the damping coefficient and r=1-d. All nodes are initialized with a PageRank Score of $\frac{1}{|V|}$.

Structural features are derived from graph structure analysis. This paper adopts Random Walk Embedding (Node2Vec). For v_i, random walk sampling generates walk sequences, and the Skip-Gram model learns low-dimensional embedding $SG(v_i)$ to capture nodes' community role similarity. The dimension of $SG(v_i)$ is d_{SG}.

The initial feature vector $h_i^{(0)}$ of v_i is obtained by concatenating the statistical features and structural features: $h_i^{(0)} = [\deg(v_i), \mathrm{CC}(v_i), \mathrm{PR}(v_i), SG(v_i)]$. The dimension of $h_i^{(0)}$ is $d = 3 + d_{SG}$. For labeled graph $G(V,E,Y)$, constructing initial features for all vertices yields $G(V,E,Y,H)$, where $h_i^{(0)} \in H$.

D. Objective Function for Balanced Graph Partitioning

Balanced partitioning aims to minimize inter-partition cut edges while ensuring approximate equal vertex counts across partitions, formally:

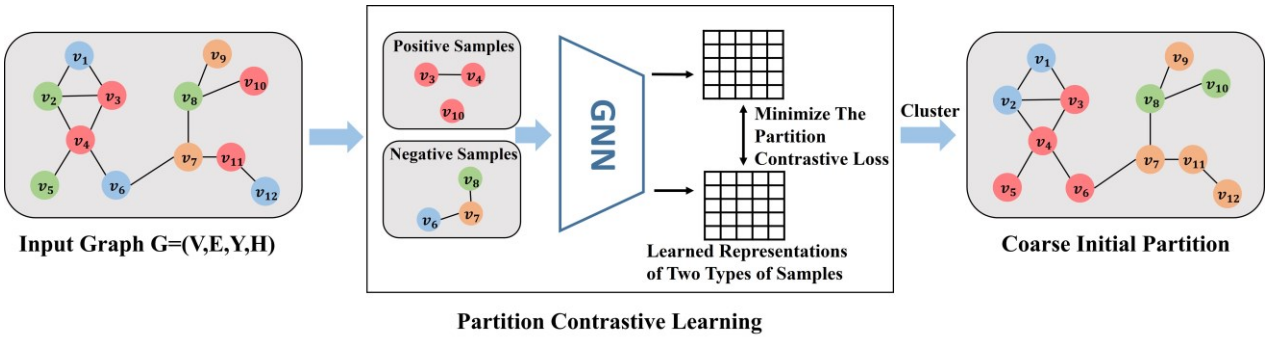

Fig. 1. Processing Framework for Generating Initial Partition Results: Input the graph G. GNN aggregates vertex features, and partition-based contrastive learning (positive/negative samples from same/different partitions) learns label-agnostic features. Clustering generates coarse initial partitions.

$$\begin{cases} \min \sum_{1\le s<t\le k} |cut(P_s,P_t)|, \\ \left|\frac{k|V_i|}{|V|}-1\right| \le \delta, \quad 1\le i\le k \end{cases} \tag{1}$$

where δ denotes the constraint relaxation parameter for the number of vertices in partitions. To simplify the problem, the penalty function method is adopted to convert the aforementioned constrained problem into an unconstrained one:

$$\min O(P) = \sum_{1\le s<t\le k} |cut(P_s,P_t)| + \alpha \sum_{i=1}^{k} \max\left(0, \left|\frac{k|V_i|}{|V|}-1\right| - \delta\right) \tag{2}$$

Here, $O(P)$ acts as both the penalty function and objective function. α is a sufficiently large positive number called penalty factor, which controls the intensity of the penalty: significant deviation from balanced partition size incurs a large penalty term, while the term is zero when balance constraints are satisfied, reducing $O(P)$ to a pure minimum cut problem. Our ultimate goal is to find the partition P that minimizes $O(P)$.

IV. Generating Initial Graph Partition Results Using Contrastive Learning

A. Processing Framework for Generating Initial Partition Results

This section proposes a contrastive learning-based method for initial partition generation. The process is as follows: first, a 3-layer graph attention network (GAT) is used to convolve the features of the input graph. Then, positive and negative samples are generated based on vertex partition labels, and network weights are updated through contrastive learning. Finally, K-Means clusters GAT-extracted vertex features to obtain the coarse initial partition. The processing framework for generating initial partition results is shown in Fig.1.

B. Embedding Representation of Vertices in Graph Neural Networks

We construct a 3-layer GAT to propagate and aggregate vertex initial features. For labeled graph $G(V,E,Y,H)$ (Y: partition labels, H: vertex initial features, $h_i^{(0)} \in H$), $h_i^{(0)}$ is input into GAT for feature propagation and aggregation. In the propagation and aggregation process of the l-th layer: First, parameterized mapping is performed on the features of all vertices through linear transformation:

$$z_i^{(l)} = W^{(l)} h_i^{(l-1)} \tag{3}$$

Here, $z_i^{(l)}$ denotes the embedding representation of vertex v_i obtained through linear transformation after convolution in the l-th layer of the network, and $W^{(l)} \in R^{d_l \times d_{l-1}}$ is a learnable weight matrix. Then, using $z_i^{(l)}$, the attention weight $c_{ij}^{(l)}$ of the adjacent vertex pair (v_i, v_j) is calculated:

$$c_{ij}^{(l)} = LeakyRelu\left((a^{(l)})^T \left[z_i^{(l)} \middle\| z_j^{(l)}\right]\right) \tag{4}$$

Where $a^{(l)} \in R^{2d_l}$ represents the learnable parameter vector of the attention mechanism, and $\|$ denotes the vector concatenation operation. Finally, the attention coefficients $\alpha_{ij}^{(l)}$ between vertex v_i and its adjacent vertices are normalized by Softmax. The neighbor features are weighted and aggregated using the normalized attention coefficients $\alpha_{ij}^{(l)}$ to obtain the updated feature $h_i^{(l)}$ of v_i:

$$\alpha_{ij}^{(l)} = \frac{\exp\left(c_{ij}^{(l)}\right)}{\sum_{k\in N(v_i)} \exp\left(c_{ik}^{(l)}\right)} \tag{5}$$

$$h_i^{(l)} = \sigma\left(\sum_{j\in N(v_i)} \alpha_{ij}^{(l)} z_j^{(l)}\right) \tag{6}$$

Here, $\sigma(\cdot)$ is a nonlinear activation function. After the graph G is convolved by the 3-layer graph attention network, the feature $h_i^{(3)}$ of vertex v_i is output. These features are then fed into contrastive learning.

C. Contrastive Learning of Vertices in Partition Blocks

We train the graph neural network using contrastive learning to meet the permutability requirement of the graphpartitioning task. For any permutation operation F on arbitrary elements in a finite set A, if the new set after permutation satisfies $F(A) = A$, then set A is said to have permutability. Graph partitioning divides the graph G into k subsets, resulting in the partition set $P = \{P_1, P_2, \ldots, P_k\}$. Swapping the positions of any two partitions in P leaves P unchanged, so P also has permutability. Therefore, if the numbers of any two partitions in the graph training data are swapped, the partition result remains unchanged, and the model should learn the same partitioning strategy (i.e., the same model weights). Contrastive learning increases the similarity of vertices in the same partition and reduces the similarity of vertices in different partitions during training, eliminating the interference of partition numbers, enabling the model to learn partition-independent features and meet the permutability requirement of the task.

1) Sampling Method of Positive and Negative Samples

Contrastive learning relies on positive and negative samples. We define vertex pairs in the same partition as positive sample pairs and vertex pairs in different partitions as

negative sample pairs, forcing the model to enhance the similarity of vertices in the same partition and reduce the similarity of cross-partition vertices in the embedding space. The specific steps are as follows: given an anchor point v_i, positive sample pairs (v_i, v_j) are sampled from the same partition, and negative sample pairs (v_i, v_k) are sampled from different partitions based on the previously obtained pseudo-label partition information. After obtaining the positive and negative samples, a loss function needs to be constructed to guide model training.

2) Loss Function for Contrastive Learning of Partition Blocks

The loss function for partition block contrastive learning is a customized variant of InfoNCE loss for graph partitioning tasks. Based on the features of positive and negative samples after GNN convolution, the normalized similarity loss is calculated:

$$L = -\sum_{j} \log \frac{\exp(sim(h_i^{(3)}, h_j^{(3)})/\tau)}{\exp(sim(h_i^{(3)}, h_j^{(3)})/\tau) + \sum_k \exp(sim(h_i^{(3)}, h_k^{(3)})/\tau)} \quad (7)$$

Where sim denotes cosine similarity, and τ is the temperature coefficient. The purpose of this loss function is to maximize the similarity of vertices within the same partition while minimizing the similarity of vertices across different partitions. Since the goal of contrastive learning is the similarity relationship between vertex features of different partitions rather than fixed numbers, even if the partition numbers in the graph training data are permuted, the model can still learn the same weights as before the permutation, satisfying the permutability requirement of the graph partitioning task.

D. Generating Initial Partition Results

After the training of the GAT, the network weights w are obtained. For a graph G to be partitioned, inputting the graph into the trained network yields the embedding representation of each vertex. These embedding representations of all vertices can be used as the raw data for graph clustering. In this paper, K-Means is used to obtain the k-clustering $P_1, P_2, \ldots, P_k$ of graph G, which serves as the initial partition $P = \{P_1, P_2, \ldots, P_k\}$. K-Means is inherently optimized for minimizing intra-cluster feature distances rather than satisfying the dual objectives of balanced graph partitioning. As a result, the generated clusters often exhibit significant differences in the number of vertices in each cluster, directly violating the strict balance constraint that requires each partition to contain a roughly equal number of vertices. Moreover, the clustering process only leverages vertex embedding similarity while ignoring the topological connectivity of the original graph, leading to an excessive number of cut edges between partitions. Since the clustering results have poor quality as balanced partition results, further refinement is performed.

V. Reinforcement Learning-Driven Fine-Tuning of Target Partitions

A. Working Process of Fine-Tuning

The core idea of fine-tuning is pairwise optimization of adjacent partitions using reinforcement learning (RL), with global optimality achieved through multi-round iteration. The flowchart of the fine-tuning process is shown in Fig.2. For the initial k partitions derived from GNN-based coarse partitioning, we first screen adjacent pairs with non-empty cut edges to form the fine-tuning set $M = \{(P_i, P_j) | P_i \in P, P_j \in P, cut(P_i, P_j) \neq \emptyset\}$. Each pair in M is processed in random order, and the optimization process is modeled as a Markov Decision Process (MDP); the RL agent decides whether to retain or migrate boundary vertices to minimize cut edges while maintaining partition balance. After completing one round of fine-tuning for all pairs in M, we calculate the global objective function and repeat the process until convergence or the maximum number of iterations is reached.

B. Reinforcement Learning Fine-Tuning Between Two Adjacent Partitions

1) Markov Decision Process (MDP) Design

a) Objective Function

The objective function for partition pair fine-tuning aligns with the global partitioning goal, formally defined as:

$$\min O(P_s, P_t) = |cut(P_s, P_t)| + \alpha \sum_{i \in I, I = \{s,t\}} \max\left(0, \left|\frac{|k|V_i|}{|V|} - 1\right| - \delta\right) \quad (8)$$

where $|cut(P_s, P_t)|$ denotes the number of cut edges between the partition pair, and the second term is the balance penalty term.

b) State Space S

The state space eliminates redundancy by focusing on key decision units—boundary vertices and their associated partitions—with a fixed dimension independent of k. It is formally expressed as: $S = \{(P_s, P_t, v_i) | \exists e_{ij} \in \partial(P_s, P_t))\}$, where v_i is a boundary vertex of the partition pair (P_s, P_t).

c) Action Space A

The action space A is designed as a binary decision to avoid coupling with k, defined as: $A = \{a_0, a_1\}$. where a_0 represents retaining vertex v_i in the source partition P_s, and a_1 represents migrating it to the target partition P_t.

d) Reward Function r_t

The reward function correlates agent actions with the objective function, guiding the selection of actions that reduce cut edges and improve partition balance. For v_i's migration action a_1, the expected reward is defined as: $MoveExpectedReward = O(P_s, P_t) - O(P_s', P_t')$, where $O(P_s, P_t)$ and $O(P_s', P_t')$ represents the objective function values of the partition pair before and after executing migration action, The reward function is formulated as:

$$r_t = \begin{cases} -MoveExpectedReward, & a = a_0 \\ MoveExpectedReward, & a = a_1 \end{cases} \quad (9)$$

A positive $MoveExpectedReward$ (i.e., reduced objective value) yields a positive reward for a_1 to encourage migration; a negative value favors a_0 to encourage retention. This design eliminates invalid zero rewards from unchanged partition states, enabling the agent to clearly perceive decision utility.

e) State Transition P

State transition P represents deterministic state transition in our method. Each action yields a unique effect with a transition probability of 1, formally expressed as: $P(s' | s, a) = 1, \forall s \in S, a \in A$, where $s = (P_s, P_t, v_i)$ is the current state, and $s' = (P_s', P_t', v_{i+1})$ is the next state. v_{i+1} is the next boundary vertex to be adjusted. Specifically, executing action a_0 leaves P_s and P_t unchanged; executing a_1 updates P_s and P_t as well as $cut(P_s, P_t)$. After action execution,

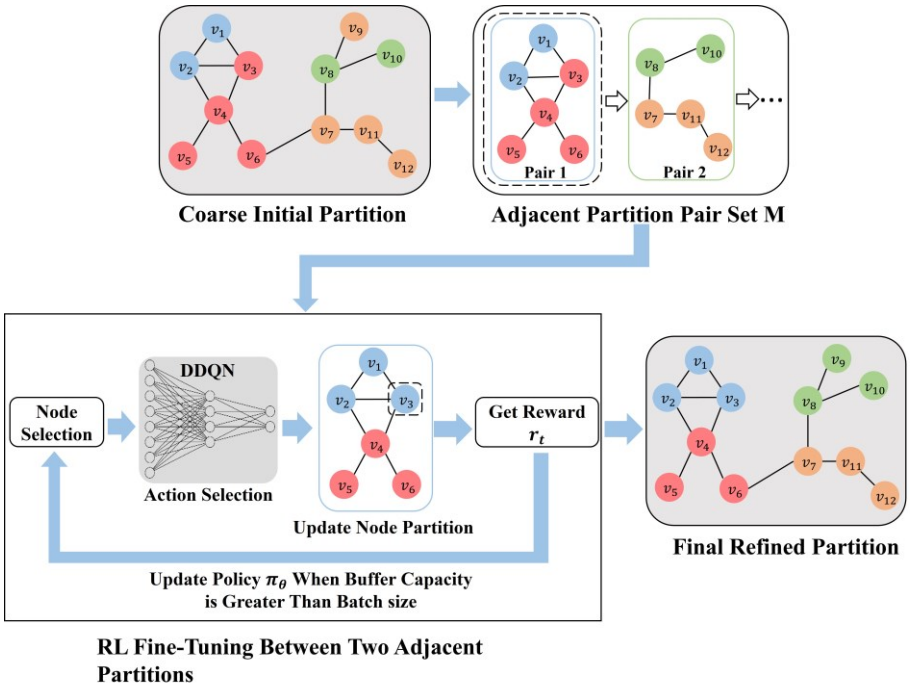

Fig. 2. Working Principle of Reinforcement Learning-based Fine-Tuning: Input coarse partitions, screen adjacent pairs with cut edges to form set M. RL models pairwise optimization as MDP and uses DDQN to adjust boundary vertices. Multi-round iteration yields refined partitions.

v_{i+1} is generated by the vertex selection module SelectNode(P_s, P_t), and the state transitions to s'. If v_{i+1} shares the same source partition as v_i, then $P_s' = P_s, P_t' = P_t$; otherwise, $P_s' = P_t, P_t' = P_s$.

The SelectNode module operates by first identifying the boundary vertex set $\text{BoundaryNode} = \{v_i | \exists e_{ij} \in cut(P_s, P_t)\}$, then computing the reduction in the number of cut edges resulting from moving each vertex in the BoundaryNode. Subsequently, it sorts these boundary vertices in descending order of cut-edge reduction, and finally randomly selects a vertex from the top-N candidates that is not in the selected vertex set Reg as v_{i+1} while adding the selected vertex v_{i+1} to the selected set Reg.

2) Feature Encoding of States

The feature vector of state s input into the policy network includes three parts:

Vertex features f_v includes local connectivity and feature similarity between the vertex and partition centers: $f_v = \left[\frac{|N(v)\cap V_s|}{|N(v)|}, \frac{|N(v)\cap V_t|}{|N(v)|}, sim(h_v, c_{P_s}), sim(h_v, c_{P_t})\right]$, where c_{P_s} denotes the source partition center, obtained by summing and averaging the vertex features of the partition.

Partition features include the source partition feature f_{P_s} and target partition feature f_{P_t}, composed of partition capacity and boundary vertex similarity within the partition:

$f_{P_s} = \left[\frac{k|V_s|}{|V|}, \frac{1}{|N(v)\cap V_s|}\sum_{u\in N(v)\cap V_s} sim(h_u, h_v)\right]$,

$f_{P_t} = \left[\frac{k|V_t|}{|V|}, \frac{1}{|N(v)\cap V_t|}\sum_{u\in N(v)\cap V_t} sim(h_u, h_v)\right]$.

Action utility features f_a calculates the immediate effect of executing the migration action, defined as:

$f_a = [\Delta_{bal}, \Delta_{cut}]$,where Δ_{bal} and Δ_{cut} represent the changes in partition imbalance and the number of cut edges after executing the vertex migration action. Finally, the input vector is formed by concatenating the above features:

$f = [f_v, f_{P_s}, f_{P_t}, f_a]$.

3) Policy Network

To optimize the vertex migration strategy, the Deep Double Q-Network (DDQN) is adopted as the core architecture of reinforcement learning. It comprises two identically structured networks with independent parameters: online network Q_θ for real-time decision-making, outputting the Q-values of each action under current state; and target network Q_{θ^-} for providing stable target Q-values.

4) Training Process

Initialize the RL fine-tuning policy network, experience replay buffer, partition pair initial state (P_s, P_t), boundary vertex set BoundaryNode and selected vertex set Reg. Select the current boundary vertex v_i through the SelectNode module, and construct the current state $s = (P_s, P_t, v_i)$. According to the feature encoding rules in Section 5.2.2, obtain the encoded vector f of state s, and feed it into the online network Q_θ to select the action a with the maximum Q-value. Calculate reward r_t after executing action a. Update P_s, P_t and $cut(P_s, P_t)$ following state transition rule P, select next vertex v_{i+1} through the SelectNode module, generate new state $s' = (P_s', P_t', v_{i+1})$, and add v_{i+1} to Reg. Store experience tuple (s, a, r, s') in the replay buffer.

When the buffer exceeds the batch size, sample a batch B to update network weights. Using DDQN strategy, calculate the optimal action $a' = argmax_{a'} Q_\theta(s', a')$ for s' through Q_θ, then calculate the target Q-value through Q_{θ^-}: $y' = r + \gamma Q_{\theta^-}(s', a')$, where γ is the discount factor. Minimize the temporal difference loss $L(\theta) = \frac{1}{|B|}\sum_{(s,a,r,s')\in B}\left(y' - Q_\theta(s,a)\right)^2$ through gradient descent to update Q_θ parameters: $\theta \leftarrow \theta - \eta\nabla_\theta L(\theta)$, where η is learning rate. Soft-update Q_{θ^-} parameters: $\theta^- \leftarrow \mu\theta + (1-\mu)\theta^-$ $(\mu \ll 1)$, ensuring that Q_{θ^-} slowly tracks the changes of Q_θ. Terminate training for the partition pair if the objective function change is less than preset threshold ϵ_{loc} or iterations reach the single partition pair training limit t_{loc}; otherwise, proceed to the next state.

C. Generating Refined Partition Results for the Entire Graph

The core idea of refined partitioning for the entire graph is to achieve global optimality via iterative local pairwise optimization. The generation of refined partition results for the entire graph mainly includes three steps:

1) Initialization and Partition Pair Screening

Input the GNN-generated initial partition results $P = \{P_1, P_2, \ldots, P_k\}$, calculate cut edge set $cut(P_i, P_j)$ for all partition pairs, and screen adjacent partition pairs with a non-empty cut edge set to form the fine-tuning set $M = \{(P_i, P_j) | P_i \in P, P_j \in P, cut(P_i, P_j) \neq \emptyset\}$.

2) Pairwise Fine-Tuning

Shuffle M to avoid order bias, and traverse each $(P_s, P_t) \in M$. For the boundary vertex set $BoundaryNode$ of (P_s, P_t), sequentially execute decisions for each vertex via the RL network, and update P_s, P_t as well as $cut(P_s, P_t)$.

3) Multiple Rounds of Iteration

After completing one round of traversal, calculate the global objective function value $O(P)$. If the change in the objective function value is less than the preset threshold ϵ_{glob} or iterations reach the preset limit t_{glob}, stop fine-tuning; otherwise, re-screen the updated fine-tuning set M and repeat pairwise fine-tuning. Integrate all partitions after convergence to obtain the final refined result $P^* = \{P_1^*, P_2^*, \ldots, P_k^*\}$.

Algorithm Expression

Algorithm 1: FullGraphRefine

Input: Initial partition $P = \{P_1, P_2, ..., P_k\}$, graph $G = (V, E)$, convergence threshold ϵ_{glob}, max iterations t_{glob}, RL policy network π

Output: Refined optimal partition $P^* = \{P_1^*, P_2^*, ..., P_k^*\}$

Initialize global state
$P_{curr} = P$; *iter* = 0; *converged* = False
Compute initial objective *prev_obj* = O(P_{curr})
while *iter* < t_{glob} **and** not *converged* **do**
    Step 1: Screen partition pairs with cut edges
    for each (P_s, P_t) in P_{curr} ($s < t$) **do**
        if *cut*(P_s, P_t) $\neq \emptyset$ **then**
            Add (P_s, P_t) to *M*
        end if
    end for
    Step 2: Pairwise Fine-Tuning
    Shuffle M to avoid order bias
    for each (P_s, P_t) **in** *M* **do**
        Initial *BoundaryNode* and *Reg*
        while *BoundaryNode**Reg* $\neq \emptyset$ **do**
            Select *v* from *BoundaryNode**Reg*
            Add *v* to *Reg*
            Decide action *a* = π(*state_features*)
            Update P_s, P_t and *cut*(P_s, P_t)
        end while
        Update $P_{curr}[s] = P_s$, $P_{curr}[t] = P_t$
    end for
    Step 3: Check global convergence
    Compute current objective *curr_obj* = O(P_{curr})
    if | *curr_obj* − *prev_obj* | < ϵ_{glob} **then**
        Set *converged* = True
    end if
    Update *prev_obj* = *curr_obj*, *iter* += 1
end while
return $P^* = P_{curr}$

D. Computational Complexity Analysis

This section analyzes the computational complexity of the DMBGP method, covering three stages: contrastive learning-based coarse partitioning, single partition-pair fine-tuning, and full partition-pair fine-tuning.

In the contrastive learning coarse partitioning stage, a 3-layer GAT is used. Let n be the number of vertices, m the number of edges, d_i the initial feature dimension, d_o the output feature dimension, and L_1 the number of layers. The complexity of node feature projection is $O(nd_id_o)$, and the complexity of attention computation is $O(md_o)$. Thus, the total inference complexity of GAT is $O(L_1(nd_id_o + md_o))$. In the single partition-pair fine-tuning stage, the DDQN adopts fully connected layers. Let p be the number of boundary vertices, d_s the state feature dimension, d_h the output dimension, and L_2 the number of network layers. The decision complexity for a single boundary vertex is $O(L_2d_sd_h)$. The decision complexity for all boundary vertices is $O(pL_2d_sd_h)$. In the full-graph fine-tuning stage, let t_{glob} be the number of global iterations and M the number of adjacent partition pairs. The total complexity is $O(t_{glob}M(pL_2d_sd_h))$. In real world graphs, M is much smaller than the total number of partition pairs k(k-1)/2. Since local optimization is only performed on boundary vertices, the complexity does not grow quadratically with k but remains approximately linear, indicating favorable scalability.

VI. EXPERIMENT

We evaluate the proposed method through extensive experiments. After describing the experimental setup, we conduct comparison experiments and analyze the results. Our code and datasets are available at https://github.com/KING-YQ-001/DMBGP.

A. Experimental Environment and Setup

All experiments are conducted under a standard computing environment. Implementations are developed with Python 3.6.4 on the Anaconda platform, using PyTorch as the deep learning framework. Contrastive learning uses a 3-layer GAT with an input dimension of 35, learning rate 0.01, and temperature τ=0.1. Node features are constructed by concatenating statistical encodings and structural encodings (32-dim Node2Vec). RL fine-tuning is based on DDQN with state dimension 10, learning rate 0.001, and γ=0.99. The balance penalty α scales with graph size, and the maximum value used in our experiments is 100000. We set δ=0.05 and perform 4 global tuning iterations.

Datasets are from SNAP [29] and Network Repository [30]. Owing to limited public attribute-less graphs with high-quality partition labels, we construct a training dataset of 40 representative samples with reliable labels from classic partitioning algorithms. Detailed parameters of the test set are shown in the following table:

Dataset	|V|	|E|	Maximum Degree	Minimum Degree	Average Degree	Local Clustering Coefficient	Global Clustering Coefficient
Reptilia-Tortoise-Network-Fi-2011	437	479	11	1	2	0.25	0.41
Mammalia-Voles-Kcs-Trapping	1218	4258	59	1	6	0.54	0.32
American75	6386	217662	930	1	68	0.24	0.16

fb-pages-sport	13866	86858	468	1	12	0.28	0.13
fb-pages-media	27917	206259	678	1	14	0.30	0.11

We adopt five core metrics to assess partitioning performance, with definitions and formulas as follows:

Cut Ratio: The ratio of total inter-partition cut edges to the graph's total edges, measuring partition connectivity. $CutRatio = \frac{\sum_{1 \le s < t \le k} |cut(P_s, P_t)|}{|E|}$.

Imbalance: The ratio of standard deviation of partition sizes to average partition size. $Imbalance = \frac{\sqrt{\frac{1}{k}\sum_{i=1}^{k}(|P_i| - |V|/k)^2}}{|V|/k}$.

Balanced Cut: A composite metric integrating Cut Ratio and Imbalance, reflecting both connectivity and balance. $BalancedCut = CutRatio + Imbalance$.

Normalized Cut: Sum of the ratios of each partition's cut edges to its total connected edges, avoiding bias from partition size. $NormalizedCut = \sum_{i=1}^{k} \frac{cut(P_i, \overline{P_i})}{vol(P_i, V)}$, where $\overline{P_i}$ denotes the complement of P_i, and $vol(P_i, V)$ is the sum of degrees of all vertices in P_i.

Modularity: Measures the difference between the actual edge density within communities and the expected edge density in a random graph. A larger Modularity indicates more distinct community structures:

$Modularity = \frac{1}{2|E|}\sum_{i,j}\left(A_{ij} - \frac{\deg(v_i) \times \deg(v_j)}{2|E|}\right)\delta(c_i, c_j)$, where A_{ij} is the adjacency matrix element, and $\delta(c_i, c_j) = 1$ if v_i, v_j belong to the same partition, otherwise 0.

Comparative experimental methods include traditional heuristic methods (METIS, KaHIP, SCOTCH, NE, K-means) and neural network methods (DMoN, MINCUTPOOL, GAP).

B. Experimental Result Analysis

We compared methods on test sets with k=2, 4 (training values) and 8 (unseen, to verify generalization). Experimental results show DMBGP achieves the best performance across all k values, demonstrating its insensitivity to k.

TABLE I. EXPERIMENTAL RESULTS ON REPTILIA-TORTOISE-NETWORK-FI-2011 DATASET

K	Method	CutRatio	Imbalance	BalancedCut	NormalizedCut	Modularity
K = 2	DMoN	0.010	0.034	0.045	0.011	0.465
	MINCUTPOOL	0.031	0.116	0.148	0.032	0.454
	GAP	0.033	0.139	0.172	0.034	0.452
	K-means	0.041	0.208	0.249	0.043	0.434
	NE	0.194	**0.002**	0.196	0.201	0.286
	SCOTCH	0.023	0.021	0.044	0.024	0.463
	KaHIP	0.022	0.020	0.043	0.023	0.463
	METIS	0.035	**0.002**	0.037	0.036	0.447
	DMBGP	**0.002**	0.021	**0.023**	**0.002**	**0.483**
K = 4	DMoN	0.104	0.004	0.108	0.104	0.621
	MINCUTPOOL	0.075	0.008	0.083	0.071	0.659
	GAP	0.063	0.147	0.210	0.064	0.664
	K-means	0.511	0.328	0.839	0.510	0.227
	NE	0.303	0.004	0.307	0.300	0.437
	SCOTCH	0.025	0.014	0.039	0.022	0.717
	KaHIP	0.019	0.021	0.040	0.017	0.726
	METIS	0.019	0.018	0.036	0.017	0.725
	DMBGP	**0.013**	**0.004**	**0.016**	**0.011**	**0.731**
K = 8	DMoN	0.157	0.047	0.203	0.147	0.713
	MINCUTPOOL	0.106	0.009	0.115	0.104	0.757
	GAP	0.129	0.053	0.183	0.118	0.725
	K-means	0.708	0.758	1.466	0.727	0.153
	NE	0.344	0.016	0.360	0.348	0.524
	SCOTCH	0.025	0.029	0.054	0.024	0.835
	KaHIP	0.027	0.040	0.067	0.025	0.841
	METIS	0.021	0.020	0.041	0.021	0.836
	DMBGP	**0.010**	**0.009**	**0.019**	**0.009**	**0.848**

TABLE II. EXPERIMENTAL RESULTS ON MAMMALIA-VOLES-KCS-TRAPPING DATASET

K	Method	CutRatio	Imbalance	BalancedCut	NormalizedCut	Modularity
K = 2	DMoN	0.112	0.051	0.163	0.113	0.386
	MINCUTPOOL	0.122	**0.000**	0.122	0.122	0.377
	GAP	0.080	0.144	0.224	0.082	0.409
	K-means	0.356	0.241	0.597	0.368	0.127
	NE	0.111	0.041	0.152	0.112	0.386
	SCOTCH	0.032	0.007	0.039	0.033	0.453
	KaHIP	0.026	0.036	0.062	0.028	0.445
	METIS	0.037	**0.000**	0.037	0.038	0.447
	DMBGP	**0.022**	**0.000**	**0.022**	**0.023**	**0.454**
K = 4	DMoN	0.193	**0.002**	0.195	0.192	0.556
	MINCUTPOOL	0.183	**0.002**	0.185	0.192	0.558
	GAP	0.142	0.065	0.207	0.150	0.591
	K-means	0.619	0.296	0.915	0.620	0.113
	NE	0.195	0.079	0.273	0.194	0.552
	SCOTCH	0.094	0.023	0.118	0.091	0.641
	KaHIP	**0.057**	0.028	0.084	**0.054**	0.676
	METIS	0.061	0.023	0.083	0.058	0.675
	DMBGP	0.058	0.020	**0.077**	0.056	**0.677**
K = 8	DMoN	0.205	0.464	0.282	0.590	0.205
	MINCUTPOOL	0.053	0.339	0.286	0.584	0.053
	GAP	0.096	0.295	0.199	0.665	0.096
	K-means	0.784	0.361	1.145	0.782	0.079
	NE	0.288	0.124	0.412	0.270	0.579
	SCOTCH	0.119	0.034	0.153	0.111	0.749
	KaHIP	0.114	0.028	0.142	0.106	0.752
	METIS	0.122	0.018	0.140	0.115	0.744
	DMBGP	**0.106**	**0.012**	**0.118**	**0.100**	**0.761**

TABLE III. EXPERIMENTAL RESULTS ON AMERICAN75 DATASET

K	Method	CutRatio	Imbalance	BalancedCut	NormalizedCut	Modularity
K = 2	DMoN	0.429	0.198	0.626	0.444	0.054
	MINCUTPOOL	0.429	0.178	0.607	0.445	0.053
	GAP	0.367	0.177	0.545	0.369	0.129
	K-means	0.336	0.065	0.401	0.439	0.046
	NE	0.385	**0.015**	0.400	0.448	0.043
	SCOTCH	0.224	0.030	0.254	0.234	0.256
	KaHIP	0.227	0.029	0.257	0.231	0.264
	METIS	0.225	0.025	0.251	0.234	0.255
	DMBGP	**0.213**	0.027	**0.240**	**0.222**	**0.266**
K = 4	DMoN	0.621	0.216	0.838	0.638	0.107
	MINCUTPOOL	0.546	0.221	0.767	0.612	0.156
	GAP	0.523	0.244	0.767	0.491	0.201
	K-means	0.653	0.247	0.901	0.688	0.052
	NE	0.637	0.021	0.659	0.681	0.065
	SCOTCH	0.345	0.041	0.386	0.333	0.391
	KaHIP	0.346	0.029	0.375	0.345	0.389
	METIS	**0.325**	0.029	0.354	0.319	0.409
	DMBGP	**0.325**	**0.020**	**0.344**	**0.318**	**0.410**
K = 8	DMoN	0.667	0.395	1.063	0.665	0.196
	MINCUTPOOL	0.785	0.179	0.964	0.807	0.058
	GAP	0.689	0.116	0.805	0.696	0.160
	K-means	0.811	0.534	1.346	0.824	0.038
	NE	0.778	**0.024**	0.802	0.804	0.072
	SCOTCH	0.512	0.050	0.562	0.517	0.354
	KaHIP	0.496	0.026	0.521	0.504	0.369
	METIS	0.490	0.030	0.519	0.514	0.370
	DMBGP	**0.464**	0.046	**0.510**	**0.494**	**0.390**

TABLE IV. EXPERIMENTAL RESULTS ON FB-PAGES-SPORT DATASET

K	Method	CutRatio	Imbalance	BalancedCut	NormalizedCut	Modularity
K = 2	DMoN	0.221	0.020	0.241	0.236	0.247
	MINCUTPOOL	0.172	0.013	0.185	0.173	0.322
	GAP	0.139	0.044	0.183	0.139	0.359
	K-means	0.423	0.480	0.904	0.425	0.073
	NE	0.327	0.023	0.351	0.425	0.056
	SCOTCH	0.101	0.027	0.128	0.101	0.398
	KaHIP	0.121	0.001	0.122	0.122	0.378
	METIS	0.098	**0.000**	0.099	0.098	0.401
	DMBGP	**0.076**	**0.000**	**0.076**	**0.076**	**0.423**
K = 4	DMoN	0.348	0.157	0.505	0.400	0.379
	MINCUTPOOL	0.358	0.079	0.437	0.421	0.346
	GAP	0.343	**0.010**	0.353	0.370	0.377
	K-means	0.620	0.290	0.911	0.660	0.077
	NE	0.554	0.030	0.584	0.633	0.114
	SCOTCH	0.147	0.021	0.168	0.148	0.602
	KaHIP	0.145	0.018	0.163	0.146	0.604
	METIS	0.149	0.030	0.179	0.150	0.600
	DMBGP	**0.121**	0.029	**0.149**	**0.122**	**0.628**
K = 8	DMoN	0.387	0.030	0.417	0.453	0.467
	MINCUTPOOL	0.467	**0.007**	0.474	0.521	0.387
	GAP	0.428	0.054	0.482	0.430	0.441
	K-means	0.771	0.395	1.166	0.793	0.079
	NE	0.696	0.035	0.732	0.747	0.134
	SCOTCH	0.204	0.037	0.241	0.203	0.670
	KaHIP	0.215	0.024	0.239	0.215	0.658
	METIS	0.212	0.028	0.240	0.215	0.660
	DMBGP	**0.196**	0.025	**0.221**	**0.199**	**0.677**

TABLE V. EXPERIMENTAL RESULTS ON FB-PAGES-MEDIA DATASET

K	Method	CutRatio	Imbalance	BalancedCut	NormalizedCut	Modularity
K = 2	DMoN	0.280	0.032	0.313	0.370	0.098
	MINCUTPOOL	0.156	0.020	0.176	0.191	0.251
	GAP	0.095	0.050	0.145	0.117	0.310
	K-means	0.270	0.080	0.350	0.362	0.102
	NE	0.283	**0.016**	0.299	0.388	0.080
	SCOTCH	0.101	0.031	0.132	0.107	0.373

	KaHIP	0.094	0.028	0.123	0.106	0.350
	METIS	0.101	0.030	0.131	0.106	0.373
	DMBGP	**0.074**	0.028	**0.103**	**0.079**	**0.397**
K = 4	DMoN	0.375	0.197	0.572	0.434	0.311
	MINCUTPOOL	0.461	0.058	0.519	0.506	0.255
	GAP	0.262	0.040	0.302	0.282	0.413
	K-means	0.592	0.298	0.890	0.628	0.094
	NE	0.531	**0.022**	0.553	0.609	0.126
	SCOTCH	0.209	0.042	0.251	0.201	0.522
	KaHIP	0.217	0.023	0.239	0.210	0.514
	METIS	0.219	0.023	0.241	0.213	0.510
	DMBGP	**0.196**	0.031	**0.227**	**0.191**	**0.532**
K = 8	DMoN	0.465	0.050	0.515	0.502	0.380
	MINCUTPOOL	0.484	0.125	0.609	0.519	0.367
	GAP	0.459	**0.005**	0.464	0.469	0.387
	K-means	0.765	0.448	1.213	0.775	0.082
	NE	0.668	0.026	0.694	0.722	0.154
	SCOTCH	0.288	0.049	0.337	0.267	0.575
	KaHIP	0.294	0.022	0.316	0.279	0.571
	METIS	0.290	0.029	0.320	0.273	0.573
	DMBGP	**0.275**	0.025	**0.300**	**0.258**	**0.588**

Ablation Experiment: We conduct an ablation study by removing the reinforcement learning module to verify the effectiveness of multi-round pairwise fine-tuning. We compare initial partitions with RL fine-tuned results on each dataset at k=8. The comparison figures show that all five partitioning metrics are significantly improved after fine-tuning, which validates the effectiveness of our RL procedure.

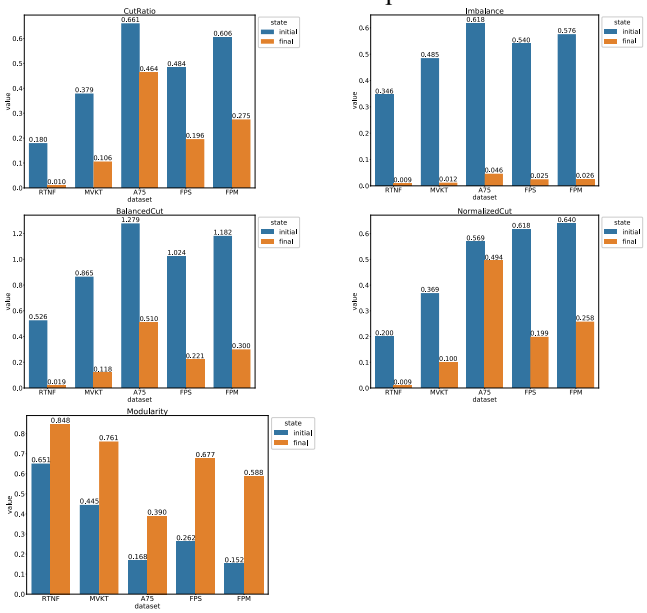

Fig. 3. Ablation results of pairwise fine-tuning on graph partitioning performance.

VII. Conclusion

This paper proposes DMBGP, a dual-mechanism balanced graph partitioning method that combines contrastive learning for coarse partitioning and reinforcement learning for fine-tuning. Specifically, contrastive learning with a customized partition loss helps the GAT learn label-agnostic features to generate initial partitions. The RL fine-tuning module only considers boundary vertices and adjacent partitions with a binary action space, which reduces redundancy, decouples the state-action space from k, and boosts efficiency. Experiments show that DMBGP outperforms traditional heuristics and advanced deep learning methods on key metrics and generalizes well to arbitrary k values.

References

[1] M. Cheng, J. Zhang, S. Long. GMaglev: Graph-friendly Consistent Hashing for Distributed Social Graph Partition. *The 2nd International Conference on Innovations and Development of Information Technologies and Robotics*, 2023, pp.30-38.

[2] A. Shalita, B. Karrer, I. Kabiljo, et al. Social hash: An assignment framework for optimizing distributed systems operations on social networks. *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, 2016, pp.455–468.

[3] D. Karger, E. Lehman, T. Leighton, et al. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing (STOC '97)*, 1997, pp.654–663.

[4] J. Lamping, E. Veach. A fast, minimal memory, consistent hash algorithm. *CoRR*, abs/1406.2294, 2014.

[5] J. P. Kim, Y. H. Kim, B. R. Moon. A Hybrid Genetic Approach for Circuit Bipartitioning. *Lecture Notes in Computer Science*, 2004, pp.1054–1064.

[6] S. Hauck, G. Borriello. An evaluation of bipartitioning techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1997, pp.849–866.

[7] A. E. Caldwell, A. B. Kahng, I. L. Markov. Improved algorithms for hypergraph bipartitioning. *Proceedings of the 2000 Asia and South Pacific Design Automation Conference (ASP-DAC '00)*, 2000, pp.661–666.

[8] B. W. Kernighan, S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 1970, pp.291–307.

[9] C. M. Fiduccia, R. M. Mattheyses. A linear-time heuristic for improving network partitions. *The 19th Design Automation Conference*, 1982, pp.175-181.

[10] D.S. Johnson, C. Aragon, L. McGeoch, et al. Optimization by Simulated Annealing: An Experimental Evaluation Part 1 Graph Partitioning. *Operations Research*, 1989, pp.865-892.

[11] G. Laszewski. Intelligent structural operators for the k-way graph partitioning problem. *Proceedings of the 4th International Conference on Genetic Algorithms*, 1991, pp.45–52.

[12] T. N. Bui, B. R. Moon. Genetic algorithm and graph partitioning. *IEEE Transactions on Computers*, 1996, 45(7):841-855.

[13] Q. Li, J. Zhong, X. Li. Implementation and Performance Optimization of Graph Partitioning in Hybrid Memory Systems. *Journal of Computer Science and Technology*, 2019, pp.2481–2498.

[14] C. Zhang, F. Wei, Q. Liu, et al. Graph edge partitioning via neighborhood heuristic, *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp.605–614.

[15] B. Hendrickson, R. W. Leland. A multi-level algorithm for partitioning graphs. *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*, 1995, 95(28): 1-14.

[16] L. Deng, N. Y. Dai, B. Xu, et al. PNFVNbM: A Metis-Based Partitioning Method for Large-Scale NFV Networks. *Journal of Computer Science and Technology*, 2019, pp.1–13.

[17] G. Karypis, V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 1998, 20(1):359-392.

[18] P. Sanders, C. Schulz. KaHIP v3.00 - Karlsruhe High Quality Partitioning User Guide. *Tech. Rep*, 2013.

[19] R. Barat, C. Chevalier, F. Pellegrini. Multi-criteria graph partitioning with Scotch. *Proceedings of the 2018 SIAM Workshop on Combinatorial Scientific Computing*, 2018, pp.66–75.

[20] W. F. Fan, R. Q. Xu, Q. Yin, et al. Application-driven graph partitioning. *The VLDB Journal*, 2023, 32(1):149-172.

[21] M. Qin, C. R. Zhang, Y. Gao et al. Towards faster graph partitioning via pre-training and inductive inference. *arXiv preprint arXiv:2409.00670*, 2024.

[22] A. Nazi, W. Hang, A. Goldie, et al. GAP: Generalizable Approximate Graph Partitioning Framework. *arXiv preprint arXiv:1903.00614v1*, 2019.

[23] F. M. Bianchi, D. Grattarola, C. Alippi. Spectral Clustering with Graph Neural Networks for Graph Pooling. *Proceedings of the 37 th International Conference on Machine Learning*, 2020, pp.1115-1124.

[24] A. Tsitsulin, J. Palowitch, B. Perozzi, et al. Graph Clustering with Graph Neural Networks[J]. *Journal of Machine Learning Research*, 2022, 23(100):1-45.

[25] H. Dai, E.B. Khalil, Y. Zhang, et al. Learning Combinatorial Optimization Algorithms over Graphs. *Proceedings of 31st Conference on Neural Information Processing Systems*, 2017, pp.6349-6359.

[26] A. C. Zhou, J. Y. Luo, R. B. Qiu, et al. Adaptive partitioning for large-scale graph analytics in geo-distributed data centers. *Proceedings of the 38th International Conference on Data Engineering*, 2022, pp.2818-2830.

[27] R. Shah, K. Jain, S. Manchanda, et al. NeuroCUT: A Neural Approach for Robust Graph Partitioning. *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024, pp.2584-2595

[28] A. Gatti, Z. Hu, T. Smidt, et al. Graph partitioning and sparse matrix ordering using reinforcement learning and graph neural networks. *Journal of Machine Learning Research*, 2022, 23(303):1-28

[29] J. Leskovec, A. Krevl. SNAP Datasets: Stanford Large Network Dataset Collection. *https://snap.stanford.edu/data*, 2014.

[30] N. S. Kaler, R. Goel, K. B. Akoglu, et al. The Network Data Repository with Interactive Graph Analytics and Visualization. *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, 2015, pp. 4292–4293.