TRADING QUALITY FOR EFFICIENCY OF GRAPH PARTITIONING: AN INDUCTIVE METHOD ACROSS GRAPHS

Anonymous authors
Paper under double-blind review

ABSTRACT

Many applications of network systems can be formulated as several NP-hard combinatorial optimization problems regarding graph partitioning (GP), e.g., modularity maximization and NCut minimization. Due to the NP-hardness, to balance the quality and efficiency of GP remains a challenge. Existing approaches try to achieve high quality via machine learning techniques but are with high complexity. Some efficient methods adopt heuristic strategies to ensure low runtime but suffer from quality degradation. In contrast to conventional transductive GP methods applied to a static graph, we propose an inductive graph partitioning (IGP) framework across multiple evolving graph snapshots to alleviate the NP-hard challenge. IGP first conducts the *offline* training of a novel dual graph neural network structure on historical snapshots to capture the structural characteristics of a system. The trained model is then generalized to newly generated snapshots for fast high-quality *online* GP without additional optimization, where a better trade-off between quality and efficiency is achieved. IGP is also a generic framework that can capture the permutation invariant partitioning ground-truth of historical snapshots in the *offline* training and handle the *online* GP on graphs with non-fixed number of nodes and clusters. Experiments on a set of benchmarks demonstrate that IGP achieves competitive quality and efficiency to various state-of-the-art baselines.

1 INTRODUCTION

For various complex systems, e.g., communication networks, graph is a generic model to describe the entities and their relations using a set of nodes and edges. Graph partitioning (GP) is a classic graph inference task that aims to partition the nodes of a graph into several groups (i.e., clusters) with dense linkage distinct from other groups. Since the extracted clusters are believed to correspond to several real-world substructures of a system, e.g., cells in wireless networks (Dai & Bai, 2017), many network applications are formulated as a GP task (Qin et al., 2019; Patil & Kulkarni, 2021).

Mathematically, GP can be described as some NP-hard combinatorial optimization problems, e.g., modularity maximization (Newman, 2006) and normalized cut (NCut) minimization (Von Luxburg, 2007). Due to the NP-hardness, to balance the quality and efficiency of GP remains a challenge, but some real applications have both the requirements of high quality and low runtime, e.g., accurate GP on a wireless cellular network with several thousand nodes in few seconds (Dai & Bai, 2017).

On the one hand, previous work has validated the potential of machine learning (ML) techniques to achieve high partitioning quality, e.g., accuracy between GP result and ground-truth. Typical ML-based approaches include non-negative matrix factorization (Wang et al., 2011) and generative probabilistic model (Karrer & Newman, 2011). Graph embedding has emerged as a promising technique for GP in recent researches. Typical graph embedding methods learn low-dimensional node representations via random walk (Perozzi et al., 2014) or deep learning (Yang et al., 2016; Wang et al., 2016) to capture high-order proximities and nonlinear characteristics of a graph. The GP result is derived by feeding the learned representations to a downstream clustering module (e.g, KMeans). However, to search for solutions to these methods, one should yield to several iterative optimization algorithms (e.g., gradient descent) to learn large-scale parameters with high complexity.

On the other hand, to reduce the runtime and satisfy the real-time constraint of some applications is another major focus. Some efficient GP methods apply heuristic strategies to approximate classic objectives, e.g., greedy modularity maximization (Blondel et al., 2008) and multi-level coarsening

for graph-cut minimization (Dhillon et al., 2007). Some efficient graph embedding approaches also adopt fast approximation to reduce the inference time, e.g., random projection for high-order proximities (Zhang et al., 2018). Despite their high efficiency (e.g., low runtime), they may still suffer from unexpected quality decline due to the inherent information loss of heuristic approximation.

Most state-of-the-art (SOTA) GP methods are inherently *transductive*, which explore the structural characteristics of a static graph and only handle the GP on such a unique graph. We try to achieve a better trade-off between quality and efficiency from a new *inductive* perspective across multiple associated graphs of a system. An inductive graph partitioning (IGP) framework is proposed based on the fact that *most real-world complex systems generate a set of snapshots via common knowledge in terms of several underlying distributions*, e.g., power-law distribution in social networks. Such a hypothesis is also widely adopted in the simulation of various network systems (Wehrle et al., 2010). IGP first trains a high-quality GP model on historical system snapshots in an *offline* way, which aims to fully capture the inherent structural properties of a system (regardless of time cost). The trained model is then generalized to newly generated snapshots for *online* GP without additional optimization, which significantly saves the runtime on these new graphs and is believed to have high-quality partitioning results. In real applications, it's usually assumed that we have enough time to prepare a high-quality model using historical data in an *offline* way, which is also one-time-effort only. Our main focus is to achieve a better trade-off between the quality and efficiency of the *online* GP on new graphs after deploying the trained model to a system (with model parameters fixed).

Technically, IGP follows a graph embedding scheme for the unsupervised node-level task (i.e., GP) across multiple graphs based on the *inductive* nature of graph neural networks (GNN). Although SOTA inductive GNNs have the potential to be generalized to new unseen nodes and graphs (Hamilton et al., 2017; Veličković et al., 2017), they still suffer from the following limitations for GP.

In this study, we consider GP on multiple graphs where topology is the only available information source, i.e., without any graph attributes. As some systems allow the addition and deletion of entities, we assume that the number of nodes N can be non-fixed for multiple graphs. Although existing inductive GNNs can be applied to graphs with non-fixed N, they're originally designed for attributed graphs and rely on the fixed dimension of node features. Our preliminary experiments indicate that some classic settings of GNN for the case without node features (e.g., use a constant matrix with fixed dimension as the feature input) may suffer from poor quality. Some methods extract features using dimension reduction (e.g., PCA (Nazi et al., 2019)) to map the topology (e.g., adjacency matrix) with non-fixed N to a feature space with fixed dimension, which is still time-consuming.

Moreover, different system snapshots may have different proper number of clusters K. Some SOTA inductive-GNN-based approaches adopt an end-to-end (E2E) framework to achieve a better approximation to classic GP objectives (e.g., GAP (Nazi et al., 2019) for NCut minimization and ClusNet (Wilder et al., 2019) for modularity maximization), with the partitioning result derived via a fully-connected (FC) output layer. Despite their high partitioning quality, they can only handle the *inductive* GP across graphs with fixed K, due to the fixed dimension of output layer. These E2E methods still need to be trained from scratch for new graphs with non-fixed K, which is time-consuming.

Although previous researches have demonstrated the potential of inductive GNNs to handle the (semi-)supervised task (e.g., node classification) on new nodes and graphs (Hamilton et al., 2017; Veličković et al., 2017), few of them consider the unsupervised node-level task (e.g., GP) across graphs. Our experiments also indicate that some standard settings of inductive GNNs for unsupervised tasks (e.g., using classic unsupervised training loss of graph embedding for GNN (Hamilton et al., 2017)) may still lack robustness for the *online* GP on newly generated graphs.

In addition to obtaining a better trade-off between quality and efficiency over conventional *transductive* approaches, IGP is also a generic framework that alleviates the aforementioned limitations of *inductive* GNNs. To handle the *online* GP with non-fixed N, we develop an efficient feature extraction strategy for *inductive* GNNs via graph coarsening. In contrast to E2E methods, IGP adopts an *inductive* graph embedding scheme across graphs, which can handle the *online* GP with non-fixed K. Note that GP is an unsupervised node-level task, where the cluster labels can be permutation invariant, e.g., label sequences $(0, 1, 1)$ and $(1, 0, 0)$ are the same in terms of GP. IGP can further incorporate such label information (e.g., GP ground-truth) of historical graphs to the *offline* training by combining classic GP objectives (e.g., modularity maximization and NCut minimization) with a novel dual GNN structure, which ensures the strong robustness for the *online* GP on new graphs.

2 PROBLEM STATEMENTS AND PRELIMINARIES

In this study, we consider GP on a set of graphs $S = \{\mathcal{G}_1, \mathcal{G}_2 \cdots, \mathcal{G}_T\}$ with snapshots from a common system. Each snapshot $\mathcal{G}_t \in S$ can be represented as a 2-tuple $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$ with $\mathcal{V}_t = \{v_1^t, \cdots, v_{N_t}^t\}$ and $\mathcal{E}_t = \{(v_i^t, v_j^t) \,|v_i^t, v_j^t \in \mathcal{V}_t\}$ denoting the set of nodes and edges. For each graph $\mathcal{G}_t$, topology is the only available information source without any graph attributes. We use the adjacency matrix $\mathbf{A}_t \in \Re^{N_t \times N_t}$ to describe its topology with N_t nodes, where $(\mathbf{A}_t)_{ij} = (\mathbf{A}_t)_{ji} = 1$ if $(v_i^t, v_j^t) \in \mathcal{E}_t$ and $(\mathbf{A}_t)_{ij} = (\mathbf{A}_t)_{ji} = 0$, otherwise. Since some systems allow the addition and deletion of entities, we assume that different snapshots in S can have different node sets, i.e., $\exists \mathcal{G}_t, \mathcal{G}_s \in S$ that $\mathcal{V}_t \neq \mathcal{V}_s$. For simplicity, the temporal dependency of S (i.e., node index correspondence among node sets $\{\mathcal{V}_1, \cdots, \mathcal{V}_T\}$) isn't considered in this study. We follow the hypothesis widely adopted in the simulation of various network systems (Wehrle et al., 2010) that *snapshots in S are independently generated via several underlying distributions of a common system*.

Graph Partitioning. Given a graph $\mathcal{G}_t$, GP aims to partition the node set $\mathcal{V}_t$ into K_t subsets (i.e., clusters) $C_t = \{C_1^t, \cdots, C_{K_t}^t\}$ so that (i) within each cluster the linkage is dense but (ii) between different clusters the linkage is relatively loose. The result C_t also satisfies $C_r^t \cap C_s^t = \emptyset$ for $\forall r \neq s$. Mathematically, GP can be formulated as several NP-hard combinatorial optimization problems, e.g., NCut minimization (Von Luxburg, 2007) and modularity maximization (Newman, 2006).

Given a graph $\mathcal{G}_t$, NCut minimization aims to get the GP result C_t that minimizes the NCut metric:

$$\arg\min_{C_t} \mathrm{NCut}(C_t) = \frac{1}{2} \sum\nolimits_{r=1}^{K_t} [w(C_r^t, \bar{C}_r^t) / \mathrm{vol}(C_r^t)], \tag{1}$$

where $\bar{C}_r^t = \mathcal{V}_t - C_r^t$ is the complementary set of C_r^t; $w(C_r^t, \bar{C}_r^t) = \sum_{v_i \in C_r^t, v_j \in \bar{C}_r^t} (\mathbf{A}_t)_{ij}$ is the cut between C_r^t and $\bar{C}_r^t$; $\mathrm{vol}(C_r^t) = \sum_{v_i, v_j \in C_r^t} (\mathbf{A}_t)_{ij}$ is the volume of C_r^t. The objective (1) can be further rewritten into the following matrix form:

$$\arg\min_{\mathbf{H}_t} \mathrm{tr}(\mathbf{H}_t^T \mathbf{L}_t \mathbf{H}_t) \ \text{ s.t. } \mathbf{H}_t^T \mathbf{H}_t = \mathbf{I}_{K_t} \tag{2}$$

where $\mathbf{L}_t = \mathbf{I}_{N_t} - \mathbf{D}_t^{-0.5} \mathbf{A}_t \mathbf{D}_t^{-0.5}$ is the Laplacian matrix of $\mathbf{A}_t$; $\mathbf{D}_t = \mathrm{diag}(d_1^t, \cdots, d_{N_t}^t)$ is a diagonal matrix with $d_i^t = \sum_j (\mathbf{A}_t)_{ij}$; $\mathbf{I}_K$ is a K-dimensional identity matrix. $\mathbf{H}_t \in \Re^{N_t \times K_t}$ is the membership indicator, where $(\mathbf{H}_t)_{ir} = [d_i^t \cdot \mathrm{vol}(C_r^t)^{-1}]^{0.5}$ if $v_i^t \in C_r^t$ and $(\mathbf{H}_t)_{ir} = 0$, otherwise.

Modularity maximization is another classic NP-hard objective of GP. Given a graph $\mathcal{G}_t$, it aims to find a partition C_t that maximizes the modularity metric:

$$\arg\max_{C_t} \mathrm{Mod}(C_t) = \frac{1}{2w} \sum\nolimits_r \sum\nolimits_{v_i^t, v_j^t \in C_r^t} [(\mathbf{A}_t)_{ij} - d_i^t d_j^t / (2w)], \tag{3}$$

where $w = \sum_i d_i^t / 2$ is the number of edges. The objective can also be rewritten into a matrix form:

$$\mathrm{argmin}_{\mathbf{H}_t} - \mathrm{tr}(\mathbf{H}_t^T \mathbf{Q}_t \mathbf{H}_t) \ \text{ s.t. } \mathrm{tr}(\mathbf{H}_t^T \mathbf{H}_t) = N_t, \tag{4}$$

where $\mathbf{Q}_t \in \Re^{N_t \times N_t}$ is the modularity matrix with $(\mathbf{Q}_t)_{ij} = (\mathbf{A}_t)_{ij} - d_i^t d_j^t / (2w)$. $\mathbf{H}_t \in \Re^{N_t \times K_t}$ is the membership indicator, where $(\mathbf{H}_t)_{ir} = 1$ if $v_i^t \in C_r^t$ and $(\mathbf{H}_t)_{ir} = 0$, otherwise.

Inductive Graph Embedding. Given a single graph $\mathcal{G}_t$, conventional *transductive* graph embedding aims to learn a function $f : \{v_i^t\} \mapsto \{\mathbf{u}_i^t \in \Re^{1 \times M}\}$ that maps each node v_i^t to an M-dimensional vector $\mathbf{u}_i^t$. The key characteristics of $\mathcal{G}_t$ (e.g., clustering structure) should be preserved in the embedding space, where the node pair (v_i^t, v_j^t) with similar properties (e.g., in the same cluster) should have similar representations $(\mathbf{u}_i^t, \mathbf{u}_j^t)$ (e.g., with close distance in the embedding space). The learned representations $\{\mathbf{u}_i^t\}$ are used as the input of several downstream tasks on $\mathcal{G}_t$ including GP.

To alleviate the NP-hard challenge of GP, we consider a novel *inductive* graph embedding scheme, where the model parameters of f are shared by all the snapshots in S. We divide S into a training set $\Gamma \subset S$ and a test set $\Gamma' = S - \Gamma$, which represent the sets of historical and newly generated snapshots, i.e., snapshots in Γ should occur before those in Γ'. We first train f on Γ in an *offline* way to fully capture the structural characteristics of historical snapshots. Although graph embedding and GP are unsupervised procedures, we assume that the partitioning result C_t of each historical graph $\mathcal{G}_t \in \Gamma$ is available in the *offline* training. Note that cluster labels in C_t are permutation invariant, e.g.,

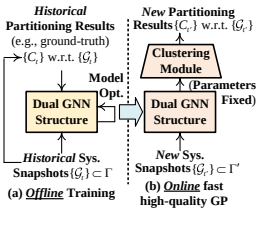


Figure 1: IGP on multiple system snapshots.

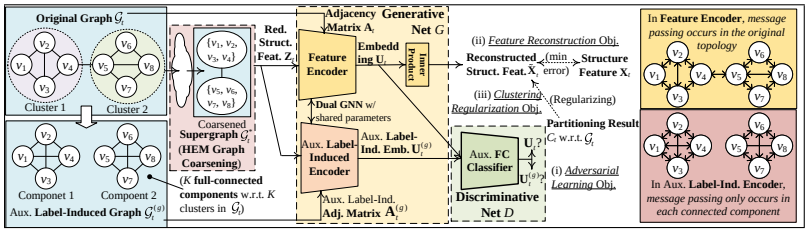


Figure 2: A running example of the *offline* training of dual GNN structure.

label sequences $(0, 1, 1)$ and $(1, 0, 0)$ are the same for GP. We consider the integration of such label information in the *offline* training. The historical partitioning results $\{C_t\}$ can be from the (i) ground-truth of system applications or (ii) results of some high-quality (but time-consuming) GP baselines, which indicates another application that we use the partitioning results of some strong baselines to regularize the *offline* training. After the *offline* training, we directly obtain the embedding $\{\mathbf{u}_i^{t'}\}$ from f for each new graph $\mathcal{G}_{t'} \in \Gamma'$ with model parameters fixed. The *online* GP on each new graph $\mathcal{G}_{t'}$ is the downstream task, where we assume that the number of clusters $K_{t'}$ is given. The partitioning result of $\mathcal{G}_{t'}$ is derived by applying a clustering algorithm (e.g., KMeans) to the embedding $\{\mathbf{u}_i^{t'}\}$.

3 METHODOLOGY

We propose a novel IGP framework with the overview shown in Fig. 1, including the (i) *offline* training of a dual GNN structure on historical graphs and (ii) *online* generalization to new graphs for fast high-quality GP via a graph embedding scheme. A simple running example (with 8 nodes and 2 clusters) regarding the *offline* training of the dual GNN structure is given in Fig. 2.

Model Architecture. Inspired by adversarial auto-encoder (AAE) (Makhzani et al., 2015), we introduce a novel dual GNN structure, including a *generative network* G and a *discriminative network* D that are jointly optimized via an adversarial process. To the best of our knowledge, the dual GNN structure is our original design, which is different from SOTA AAE-based graph embedding methods (Pan et al., 2018) that regularize the embedding via the adversarial process between a probability distribution. In contrast, our dual GNN structure incorporates the permutation invariant partitioning labels of historical graphs $\{\mathcal{G}_t\}$ to the learned embedding via (i) a parameter sharing strategy and (ii) an adversarial process between $\{\mathcal{G}_t\}$ and *auxiliary label-induced graphs* $\{\mathcal{G}_t^{(g)}\}$.

Concretely, in the *offline* training, we assume that the topology and partitioning result of each historical graph $\mathcal{G}_t$ are available, which are described by an adjacency matrix $\mathbf{A}_t \in \Re^{N_t \times N_t}$ and an *indicator matrix* $\mathbf{R}_t \in \Re^{N_t \times K_t}$. We define $(\mathbf{R}_t)_{ir} = 1$ if node v_i^t is in cluster C_r^t by the partitioning result and $(\mathbf{R}_t)_{ir} = 0$, otherwise. Due to the disjoint constraint of GP, only one entry in each row of $\mathbf{R}_t$ is 1 with other entries in the same row set to be 0. In addition to the original graph $\mathcal{G}_t$, an *auxiliary label-induced graph* $\mathcal{G}_t^{(g)}$, with $\mathbf{A}_t^{(g)} = \mathbf{R}_t\mathbf{R}_t^{\mathrm{T}}$ as the *auxiliary adjacency matrix* describing its topology, is introduced to encode the structure of partitioning result based on Theorem 1.

(**Theorem 1**) For a historical graph $\mathcal{G}_t$ with K_t clusters, its *auxiliary label-induced graph* $\mathcal{G}_t^{(g)}$ has K_t fully connected components with each component corresponding to one unique cluster of $\mathcal{G}_t$. Namely, there's an edge between nodes v_i^t and v_j^t in $\mathcal{G}_t^{(g)}$ only when they are in the same cluster, thus encoding the label information of partitioning result. In the example of Fig. 2 (with 8 nodes), $\mathcal{G}_t^{(g)}$ has 2 fully connected components $\{v_1, v_2, v_3, v_4\}$ and $\{v_5, v_6, v_7, v_8\}$ with respect to the 2 clusters in the partitioning result of $\mathcal{G}_t$. The proof of Theorem 1 is given in Appendix A.

The generative network G contains a *feature encoder* and an auxiliary *label-induced encoder*, which are GNNs with shared parameters δ_G. The *feature encoder* takes $\mathbf{A}_t$ and a feature matrix $\mathbf{X}_t$ as inputs to derive graph embedding $\mathbf{U}_t$, i.e., $\mathbf{U}_t = G(\mathbf{A}_t, \mathbf{X}_t; \delta_G)$. The *label-induced encoder* takes $\mathbf{A}_t^{(g)}$ and $\mathbf{X}_t$ as inputs, and outputs *auxiliary embedding* $\mathbf{U}_t^{(g)}$, i.e., $\mathbf{U}_t^{(g)} = G(\mathbf{A}_t^{(g)}, \mathbf{X}_t; \delta_G)$.

GNN is originally designed for attributed graphs, where each node v_i^t must have a feature vector described by the i-th row of the feature matrix $\mathbf{X}_t$. In this study, we consider GP without node

features. Instead of using the classic settings of GNN for graphs without attributes (e.g., treat a constant matrix as the feature input), we extract additional structural feature $\mathbf{X}_t$ based on the neighbor similarity encoded in $\mathbf{A}_t$. IGP captures the (permutation invariant) partitioning results of historical graphs by combing the dual GNN structure with classic GP objectives. We adopt the objectives of (i) modularity maximization and (ii) NCut minimization as two examples, corresponding to two settings of $\mathbf{X}_t$. For modularity maximization in (4), the modularity matrix $\mathbf{Q}_t$ encodes the reweighted neighbor similarity based on $\mathbf{A}_t$, so we let $\mathbf{X}_t = \mathbf{Q}_t$. For NCut minimization in (2), the Laplacian matrix $\mathbf{L}_t$ gives the primary characteristics of the graph structure, where $\mathbf{M}_t = \mathbf{D}_t^{-0.5}\mathbf{A}_t\mathbf{D}_t^{-0.5}$ is the key component regarding the neighbor similarity. Hence, we let $\mathbf{X}_t = \mathbf{M}_t$. Both $\mathbf{Q}_t$ and $\mathbf{M}_t$ are the reweighting of $\mathbf{A}_t$, where the node pair (v_i^t, v_j^t) with similar neighbor-induced feature $((\mathbf{X}_t)_{i,:}, (\mathbf{X}_t)_{j,:})$ are more likely to be partitioned into the same cluster. Our experiments further demonstrate that the extraction of $\mathbf{X}_t$ is essential to ensure the high partitioning quality of IGP.

However, the feature dimension of $\mathbf{X}_t$ is N_t (i.e., the number of nodes) which may be non-fixed. We introduce an efficient feature extraction module via the heavy-edge matching (HEM) graph coarsening (Hendrickson & Leland, 1995) to map $\mathbf{X}_t \in \Re^{N_t \times N_t}$ to another matrix $\mathbf{Z}_t \in \Re^{N_t \times L}$ with fixed feature dimension L. For $N_t > L$, we extract reweighted edges $\mathcal{E}_t^{(w)} = \{w(v_i^t, v_j^t) | w(v_i^t, v_j^t) = (\mathbf{X}_t)_{ij}, (\mathbf{A}_t)_{ij} = 1\}$ and apply HEM to $\mathcal{E}_t^{(w)}$. It merges the original graph $\mathcal{G}_t$ with N_t nodes into a supergraph $\mathcal{G}_t^*$ with L supernodes via a greedy strategy of continuously merging the node pair with largest weight in $\mathcal{E}_t^{(w)}$ into a supernode, e.g., merging a graph with 8 nodes into 2 supernodes in Fig. 2. HEM outputs a *coarsening matrix* $\mathbf{C}_t \in \Re^{N_t \times L}$, where $(\mathbf{C}_t)_{ij} = |v_j^{t*}|^{-0.5}$ if node v_i^t is merged into supernode v_j^{t*} and $(\mathbf{C}_t)_{ij} = 0$, otherwise. We then let $\mathbf{Z}_t = \mathbf{X}_t\mathbf{C}_t$ be the reduced feature. Since $\mathbf{C}_t$ is a sparse matrix, one can obtain $\mathbf{Z}_t$ by setting its j-th column as $(\mathbf{Z}_t)_{:,j} = \sum_{v_i \in v_j^{t*}} (\mathbf{C}_t)_{ij} (\mathbf{X}_t)_{:,i}$. For $N_t < L$, we directly set $\mathbf{Z}_t = [\mathbf{X}_t, \mathbf{0}_{N_t \times (L - N_t)}]$. To the best of our knowledge, using HEM to extract structural feature $\mathbf{Z}_t$ for inductive GNNs (with non-fixed node set) is our original design. It's more efficient than dimension reduction algorithms (e.g., PCA of $\mathbf{A}_t$) used in SOTA GNNs (Nazi et al., 2019), since HEM is widely used in fast GP methods (Dhillon et al., 2007). The reduced feature $\mathbf{Z}_t$ is also more informative than the feature input in some classic settings of inductive GNNs (e.g., let $\mathbf{Z}_t$ be a constant matrix (Xu et al., 2018)).

Let $\mathbf{F}_t^{(l-1)}$ and $\mathbf{F}_t^{(l)}$ be the input and output of the l-th layer of the *feature encoder* or *label-induced encoder* in G with $\mathbf{F}_t^{(0)} = \mathbf{Z}_t$. The l-th GNN encoder layer is defined as follow:

$$\mathbf{F}_t^{(l)} = \tanh(\hat{\mathbf{D}}_t^{-0.5}\hat{\mathbf{A}}_t\hat{\mathbf{D}}_t^{-0.5}\mathbf{F}_t^{(l-1)}\mathbf{W}_G^{(l-1)}), \tag{5}$$

where $\hat{\mathbf{A}}_t = \mathbf{A'}_t + \mathbf{I}_{N_t}$ ($\mathbf{A'}_t \in \{\mathbf{A}_t, \mathbf{A}_t^{(g)}\}$) is the adjacency matrix with self-edges; $\hat{\mathbf{D}}_t = \mathrm{diag}(\hat{d}_1^t, \cdots, \hat{d}_{N_t}^t)$ is the diagonal degree matrix; $\mathbf{W}_G^{(l-1)}$ is the trainable parameter shared by the two encoders; $(\mathbf{F}_t^{(l)})_{i,:}$ is the latent feature of node v_i^t. In (5), $(\mathbf{F}_t^{(l)})_{i,:}$ is the nonlinear aggregation of the features of $\{v_i^t\} \cup n(v_i^t)$ in previous layer, where $n(v_i^t)$ is the neighbor set of v_i^t. It's also known as the *message passing* of GNN, where the neighbors of v_i^t propagate their features to v_i^t for aggregation. By Theorem 1, $\mathbf{A}_t^{(g)}$ ensures that the *message passing* of *label-induced encoder* only occurs in each connected component with respect to each cluster in the partitioning result (e.g., the 2 components in Fig. 2), while the *message passing* of *feature encoder* occurs in the original topology of $\mathcal{G}_t$. The last layer of the *feature* and *label-induced encoder* output the graph embedding $\mathbf{U}_t \in \Re^{N_t \times M}$ and *auxiliary label-induced embedding* $\mathbf{U}_t^{(g)} \in \Re^{N_t \times M}$, where $\mathbf{U}_t^{(g)} = G(\mathbf{A}_t^{(g)}, \mathbf{X}_t)$ preserves more informative label-induced properties than $\mathbf{U}_t = G(\mathbf{A}_t, \mathbf{X}_t)$. We further use the non-linear inner product of $\mathbf{U}_t$ to reconstruct the neighbor-induced features $\mathbf{X}_t$:

$$\tilde{\mathbf{X}}_t = \tanh(\mathbf{U}_t\mathbf{U}_t^{\mathrm{T}}). \tag{6}$$

The discriminative network D is an auxiliary classifier that aims to distinguish $\mathbf{U}_t^{(g)}$ from $\mathbf{U}_t$, while the generative network G tries to generate plausible embedding $\mathbf{U}_t$ to fool D. Such an adversarial process helps G to generate embedding $\mathbf{U}_t$ close to $\mathbf{U}_t^{(g)}$. We denote D as $\mathbf{y}_t = D(\mathbf{S}_t; \delta_D)$, where $\mathbf{S}_t \in \{\mathbf{U}_t, \mathbf{U}_t^{(g)}\}$ and δ_D are the input and set of model parameters; $\mathbf{y}_t \in \Re^{N_t}$ is a column vector with $(\mathbf{y}_t)_i$ as probability that $(\mathbf{S}_t)_{i,:} = (\mathbf{U}_t^{(g)})_{i,:}$ rather than $(\mathbf{S}_t)_{i,:} = (\mathbf{U}_t)_{i,:}$. Let $\mathbf{P}_t^{(l-1)}$ and $\mathbf{P}_t^{(l)}$ be the input and output of the l-th layer in D with $\mathbf{P}_t^{(0)} = \mathbf{S}_t$. The l-th layer of D is defined as

$$\mathbf{P}_t^{(l)} = \mathrm{ReLU}(\mathbf{P}_t^{(l-1)}\mathbf{W}_D^{(l-1)} + \mathbf{b}_D^{(l-1)}), \tag{7}$$

where $\{\mathbf{W}_D^{(l-1)}, \mathbf{b}_D^{(l-1)}\}$ are trainable parameters. In particular, we use sigmoid as the activation function of the last layer instead of ReLU.

Model Optimization. As in Fig. 2, the *offline* training of IGP is based on three objectives of (i) adversarial learning (AL), (ii) feature reconstruction (FR), and (iii) clustering regularization (CR).

The AL objective helps incorporate the permutation invariant label information of historical graphs to the unsupervised embedding learning via an adversarial process between D and G. On the one hand, D tries to distinguish $\mathbf{U}_t^{(g)}$ from $\mathbf{U}_t$. The objective of D with respect to a graph $\mathcal{G}_t$ is

$$\arg\min_{\delta_D} L_D(\mathcal{G}_t) = -\left[\sum\nolimits_i \log(1 - D(\mathbf{U}_t)_i) + \sum\nolimits_i \log D(\mathbf{U}_t^{(g)})_i\right] / N_t. \tag{8}$$

On the other hand, G tries to derive plausible embedding $\mathbf{U}_t$ to fool D by minimizing the following loss with respect to $\mathcal{G}_t$:

$$L_{\mathrm{AL}}(\mathcal{G}_t) = -\left[\sum\nolimits_i \log D(\mathbf{U}_t)_i\right] / N_t. \tag{9}$$

Such an adversarial process directs G to output the embedding $\mathbf{U}_t$ that is close to $\mathbf{U}_t^{(g)}$, thus enabling G to capture the permutation invariant label information of historical graphs.

The FR loss forces G to derive the embedding $\mathbf{U}_t$ that encodes the key properties of the neighbor-induced feature $\mathbf{X}_t$ with respect to $\mathcal{G}_t$ by minimizing reconstruction error between $\mathbf{X}_t$ and $\tilde{\mathbf{X}}_t$:

$$L_{\mathrm{FR}}(\mathcal{G}_t) = ||\tilde{\mathbf{X}}_t - \mathbf{X}_t||_F^2. \tag{10}$$

In addition to the AL loss, IGP can also capture the (permutation invariant) label information via the regularization of classic GP objectives, e.g., modularity maximization (4) and NCut minimization (2). We introduce the CR objective with respect to a graph $\mathcal{G}_t$ that minimizes the following loss:

$$L_{\mathrm{CR}}(\mathcal{G}_t) = -\mathrm{tr}(\mathbf{H}_t^{\mathrm{T}} \tilde{\mathbf{X}}_t \mathbf{H}_t), \tag{11}$$

where $\mathbf{H}_t$ is the membership indicator encoding the partitioning result with same definitions as in (4) and (2). Since the constraints on $\mathbf{H}_t$, e.g., $\mathrm{tr}(\mathbf{H}_t^T \mathbf{H}_t) = N_t$ and $\mathbf{H}_t^T \mathbf{H}_t = \mathbf{I}_{K_t}$, are always satisfied for a given partitioning result, we don't need to consider the discrete constraints on $\mathbf{H}_t$.

Finally, we drive the objective of G with respect to $\mathcal{G}_t$ by combing the loss (9), (10), and (11):

$$\arg\min\nolimits_{\delta_G} L_G(\mathcal{G}_t) = L_{\mathrm{AL}}(\mathcal{G}_t) + \alpha L_{\mathrm{FR}}(\mathcal{G}_t) + \beta L_{\mathrm{CR}}(\mathcal{G}_t), \tag{12}$$

where $\{\alpha, \beta\}$ are parameters to balance L_{FR} and L_{CR}.

In the joint *offline* optimization of D and G, we use the Xavier algorithm to initialize model parameters $\{\delta_D, \delta_G\}$. The Adam algorithm is applied to iteratively update δ_D and δ_G based on the gradient of loss (8) and (12). In each epoch, we randomly sample a certain number of historical graphs p from the training set to update $\{\delta_D, \delta_G\}$. Finally, we save the model parameters $\{\delta_D^*, \delta_G^*\}$ that result in the best average quality on the validation set within a certain number of epochs n.

After the *offline* training, we generalize IGP to new unseen graphs for fast *online* GP. As in Fig. 1, for each new graph $\mathcal{G}_{t'}$, we derive its embedding $\mathbf{U}_{t'}$ by passing $\{\mathbf{A}_{t'}, \mathbf{X}_{t'}\}$ forward the *feature encoder* of G with its parameters δ_G fixed. As we assume that the number of clusters $K_{t'}$ is given, we apply KMeans to $\mathbf{U}_{t'}$, which outputs the partitioning result with respect to the given $K_{t'}$. Therefore, the runtime of *online* GP on $\mathcal{G}_{t'}$ includes (i) the efficient feature extraction of $\mathbf{Z}_t$, (ii) one feedforward propagation through the *feature encoder*, and (iii) the downstream clustering. Due to the space limit, we conclude the *offline* training and *online* generalization procedures of IGP in Appendix B.

4 EXPERIMENTS

Datasets. We evaluate the effectiveness of IGP on 2 synthetic benchmarks (with 7 settings) and 4 real datasets. Statistic details are depicted in Table 1, where T, N, $|\mathcal{E}|$, and K are the number of graphs, nodes, edges, and clusters. The 11 datasets cover the cases with (i) fixed N and K, (ii) fixed N but non-fixed K, as well as (iii) non-fixed N and K.

GN-Net (Girvan & Newman, 2002) and *LFR-Net* (Lancichinetti et al., 2008) are widely-used synthetic benchmarks that can simulate properties of real-world network systems. In each benchmark,

we generated a set of graphs and their partitioning ground-truth via several distributions as a simulation of real systems. *GN-Net* and *LFR-Net* use p_{in} and μ to control the clustering structures of each snapshot. We denote the *GN-Net* with a setting of p_{in} as *GN-*p_{in} ($p_{\text{in}} \in \{0.5, 0.4, 0.3\}$). The 3 datasets of *GN-Net* are with fixed N and K. We use $L(T, \mu)$ ($T \in \{f, n\}$, $\mu \in \{0.3, 0.6\}$) to represent the *LFR-Net* with a setting of μ, where f and n denote the cases with fixed and non-fixed N. *As the decrease of* p_{in} *and increase of* μ*, the clustering structures are increasingly difficult to be identified.* *Taxi* (Piorkowski et al., 2009), *Reddit* (Yanardag & Vishwanathan, 2015), *Enron* (Rossi & Ahmed, 2015), and *AS* (Leskovec et al., 2005) are real datasets from a vehicle network, an email network, a social network, and an autonomous system. As the 4 real datasets don't provide ground-truth regarding the K value of each graph, to ensure the fairness of comparison, where each graph is with a common K for all the methods, we applied the widely-used *Louvain* algorithm (Blondel et al., 2008) to determine K for each graph. Details of the datasets are elaborated in Appendix C.2.

Baselines. We compared IGP over 15 baselines with 4 categories. **(i)** *Symmetry NMF* (*SNMF*) (Wang et al., 2011) and *spectral clustering* (*SC*) (Von Luxburg, 2007) are classic *transductive* GP methods. **(ii)** *VERSE* (Tsitsulin et al., 2018), *MILE* (Liang et al., 2018), and *PhUSION* (Zhu et al., 2021) are SOTA graph embedding approaches for high-quality representation, while *NPR* (Yang et al., 2020), *RandNE* (Zhang et al., 2018), and *ProNE* (Zhang et al., 2019) are efficient embedding methods with fast approximation. All these embedding methods are *transductive*. **(iii)** *Metis* (Hendrickson & Leland, 1995), *hMetis* (Selvakkumaran & Karypis, 2006), and *GraClus* (Dhillon et al., 2007) are classic fast (*transductive*) baselines with heuristic approximation. **(iv)** *GraSAGE* (with mean aggregator) (Hamilton et al., 2017) and *GAT* (Veličković et al., 2017) are SOTA *inductive* GNNs, while *GAP* (Nazi et al., 2019) and *ClusNet* (Wilder et al., 2019) are E2E methods based on *inductive* GNNs. Besides the aforementioned methods, our comparison also includes some other strong baselines (e.g., *node2vec* (Grover & Leskovec, 2016)), with details given in Appendix C.3.

Both the variants of IGP were evaluated. For simplicity, we use IGP-M and IGP-C to denote the variants adopting the objectives of modularity maximization and NCut minimization, respectively.

Quantitative Evaluation. We consider the GP on multiple graph snapshots, where K is given for each graph. For all the datasets, we adopted the quality metrics of modularity and NCut as in (3) and (1). For datasets with ground-truth (i.e., *GN-Net* and *LFR-Net*), we also used normalized mutual information (NMI) and accuracy (AC) (Cui et al., 2018) to measure the correspondence between partitioning result and ground-truth. Usually, larger modularity, NMI and AC, as well as smaller NCut indicate better quality. Moreover, the total runtime of each method was also recorded.

Most conventional baselines (e.g., *SC*) are *transductive*, which can only be applied to a static graph, but IGP handles the NP-hard challenge of GP in an *inductive* framework across graphs. To illustrate such superiority of IGP beyond *transductive* baselines, for each dataset, we used the first 80% of the graphs as the training set Γ_{T} with the rest 10% and 10% deployed as the validation set Γ_{V} and test set Γ'. In this setting, $\Gamma_{\text{T}} \cup \Gamma_{\text{V}}$ and Γ' represent the set of historical and newly generated graphs, respectively. We conducted the *offline* training of IGP on Γ_{T} and generalized it to Γ' for *online* GP. In the *offline* training, we save the model parameters of IGP with the best average quality on Γ_{V}, where we use NMI and modularity as the validation quality metric for datasets with and without ground-truth, respectively. Since we focus on the trade-off between quality and efficiency on new graphs of a system, the evaluation was conducted on Γ', where we had to apply *transductive* baselines to each test graph in Γ' from scratch due to their *transductive* nature.

For *inductive* GNN baselines (i.e., *GraphSAGE* and *GAT*), we use the same training and evaluation settings with IGP, i.e., *offline* training on Γ_{T} and *online* generalization to Γ'. Since we consider the GP on graphs without attributes, we use a constant matrix $\mathbf{1}_{N_t \times L}$ (with fixed feature dimension L), where each entry is set to be 1, as the feature input of these GNNs, which is a widely-used setting of inductive GNNs for the case without attributes (Xu et al., 2018). In the *offline* training of *inductive* GNN baselines, we adopted the widely-used unsupervised loss of *GraSAGE* (Hamilton et al., 2017) to train them on Γ_{T}, since most existing inductive GNNs (even with a supervised loss like cross-entropy) cannot integrate of the (permutation invariant) label information from Γ_{T}.

For the E2E GNN methods (i.e., *GAP* and *ClusNet*), we could only conduct the *offline* training and fast *online* generalization on datasets with fixed K (i.e., on *GN-Net*), but we had to train them from scratch on the test set of other datasets with non-fixed K, due to the fixed dimension of their E2E output layers. As recommended in (Nazi et al., 2019) and (Wilder et al., 2019), we used *PCA* and

Table 1: Statistic details of the datasets.

	T	N	$\lvert\mathcal{E}\rvert$	K
GN-0.5	2,000	5,000	48,065-49,314	250
GN-0.4	2,000	5,000	48,335-49,698	250
GN-0.3	2,000	5,000	48,488-49,981	250
L(*f*,0.3)	2,000	5,000	22,539-25,707	57-104
L(*f*,0.6)	2,000	5,000	22,585-25,639	58-107
L(*n*,0.3)	2,000	5,000-5,999	22,999-29,670	59-117
L(*n*,0.6)	2,000	5,000-6,000	22,680-29,931	61-118
Taxi	3,000	1,279	38,554-40,171	7-12
Reddit	1,870	501-3,648	525-4,780	14-121
Enron	410	502-3,261	554-5,097	10-31
AS	733	103-6,474	487-26,467	7-39

Table 2: Ablation study on *L*(*n*,.6).

	NMI (↑)	**AC** (↑)	**Modularity** (↑)	**Ncut** (↓)
IGP-M	**0.5648**	**0.5342**	**0.3422**	**82.7605**
w/o AL	0.5213	0.4930	0.3335	84.054
w/o FR	0.1119	0.0628	0.1008	1.01e3
w/o CR	0.4997	0.4670	0.3270	86.0849
w/o $\mathbf{Z}_t$	0.1639	0.0614	0.0127	3.01e4
w/o GNN	0.0114	0.0392	0.0001	1.08e4
IGP-C	**0.5271**	**0.4747**	**0.3195**	**92.3965**
w/o AL	0.3512	0.2615	0.2670	113.0982
w/o FR	0.1546	0.0546	0.0045	2.08e4
w/o CR	0.3516	0.2622	0.2678	111.6198
w/o $\mathbf{Z}_t$	0.1388	0.0557	0.0028	2.88e4
w/o GNN	0.0845	0.0478	0.0032	5.12e4

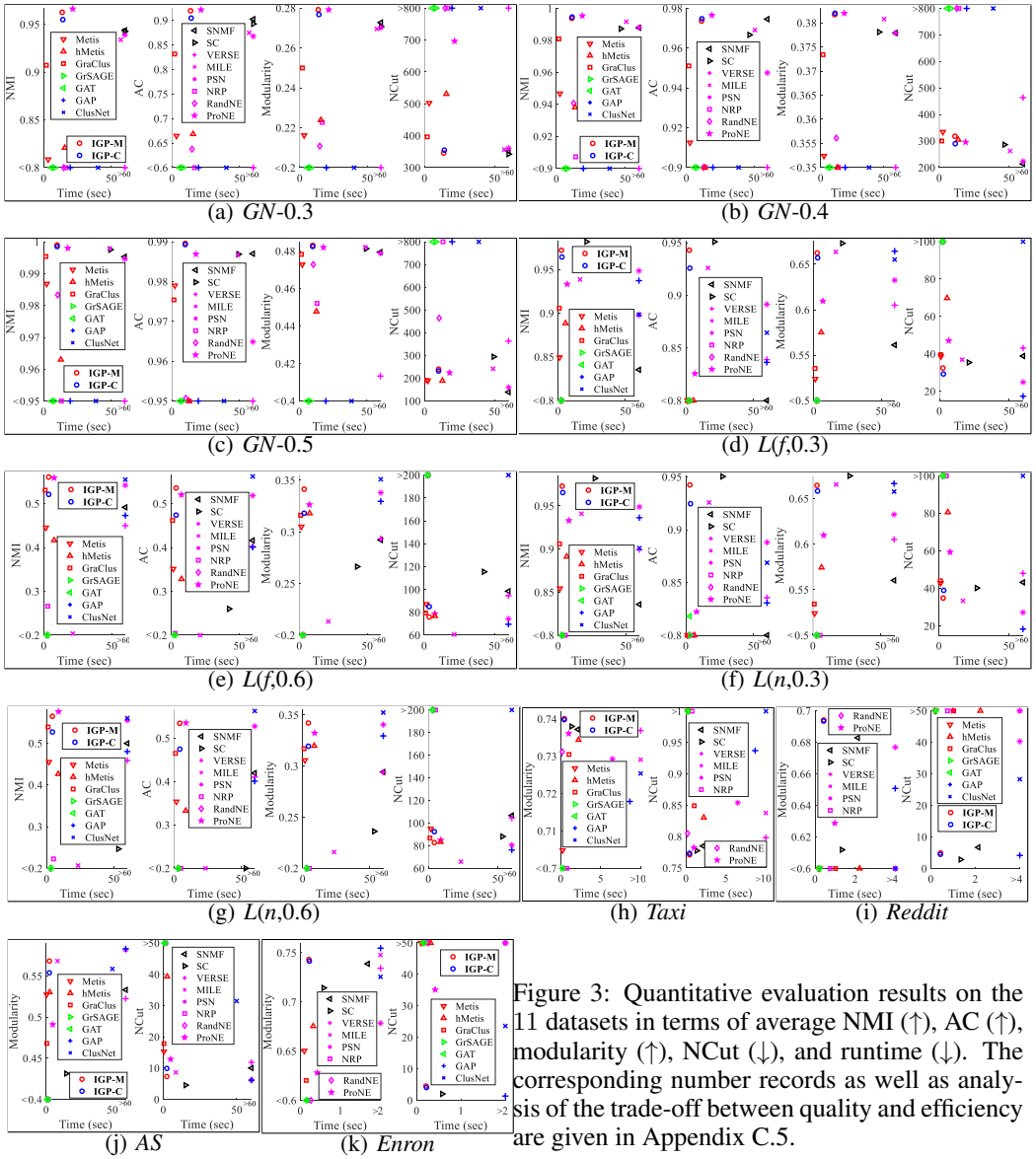
node2vec to map adjacency matrices $\{\mathbf{A}_t\}$ to a feature space with fixed dimension for the GNN feature inputs of *GAP* and *ClusNet*. Moreover, we used their own unsupervised training loss, which approximate the objectives of NCut minimization and modularity maximization, respectively.

For all the embedding-based methods (including the *transductive* embedding baselines, *inductive* GNNs, and IGP), we used KMeans as the downstream clustering module. We ran KMeans 10 times to avoid its limitation of getting local optimal solutions, which is included in the total runtime. To ensure the fairness of comparison, the embedding dimensions of all the embedding-based methods were set to be the same on each dataset. For other E2E baselines (e.g., *Metis*, *GAP* and *ClusNet*), we directly derived the partitioning results from their outputs. Since *Taxi*, *Reddit*, *Enron*, and *AS* don't provide the partitioning ground-truth (in terms of partitioning membership C_t), on each historical training snapshot of these datasets, we used the partitioning result of the baseline with best average modularity to derive the membership indicator $\mathbf{H}_t$ for IGP (i.e., the second source of historical label information defined in Section 2). Note that we don't need to derive $\mathbf{H}_t$ in the evaluation phase when generalizing the trained model to test set Γ'.

We tuned parameters of all the methods on validation set Γ_{V} of each dataset to report the best average quality metrics. Both the mean and standard deviations of all the quality metrics and runtime on test set Γ' of each dataset were recorded. Due to the space limit, we give the number records of mean and standard deviations, as well as the parameter settings in Appendix C.5 and C.4. The evaluation results in terms of average NMI, AC, modularity, NCut, and runtime are illustrated in Fig. 3. In most cases, IGP-M and IGP-C are in the top group with best quality metrics, even though we didn't train them on the test set Γ', where other *transductive* baselines were fully trained. Surprisingly, IGP even performs the best or second-best in some cases. Moreover, the runtime of IGP-M and IGP-C are also competitive to the fast GP baselines. Although the runtime of *inductive* GNN baselines (with the same fast *online* generalization setting) is slightly faster than IGP, they still suffer from poor quality, which implies that the neighbor-induced feature $\mathbf{Z}_t$ and hybrid training loss of IGP are key components to ensure its high quality. From the view of alleviating the NP-hard challenge of *online* GP on newly generated system snapshots, the *inductive* framework of IGP achieves a better trade-off between quality and efficiency. In Appendix C.5, we further quantitatively evaluate the significant trade-off of IGP beyond other baselines by introducing a trade-off score and also give some other discussions as well as analysis of the evaluation results.

Ablation Study. For both IGP-M and IGP-C, we also validated the effectiveness of the (i) AL loss, (ii) FR loss, (iii) CR loss, (iv) reduced feature input $\mathbf{Z}_t$, and (v) GNN by respectively excluding the corresponding component from the original model. In case (iv), we used a constant matrix $\mathbf{1}_{N_t \times L}$ (with all entries set to be 1) to replace $\mathbf{Z}_t$, while we use an FC network that only takes $\mathbf{Z}_t$ as the input to replace the GNN encoder of G (with the same layer configurations). Example results on $L(n, .6)$ in terms of average NMI, AC, modularity, and NCut are shown in Table 2. According to Table 2, the FR loss, feature input $\mathbf{Z}_t$, and GNN are key components to ensure the high quality of IGP, since there're significant quality declines in case (ii), (iv), and (v). The AL and CR loss further enhance the quality by incorporating the permutation invariant label information of historical snapshots. Due to the space limit, we leave the details of some other ablation studies in Appendix C.5.

Extended Applications. In addition to the aforementioned experiments, we consider two extended applications. First, when the number of clusters K of a new graph isn't given, IGP also has the potential to estimate the K value, where we apply an auxiliary model selection method (e.g., XMeans) to the derived embedding $\mathbf{U}_t$. Second, in contrast to the hypothesis that all the given snapshots are



(a) *GN*-0.3 (b) *GN*-0.4 (c) *GN*-0.5 (d) *L*(*f*,0.3) (e) *L*(*f*,0.6) (f) *L*(*n*,0.3) (g) *L*(*n*,0.6) (h) *Taxi* (i) *Reddit*

(j) *AS* (k) *Enron*

Figure 3: Quantitative evaluation results on the 11 datasets in terms of average NMI (↑), AC (↑), modularity (↑), NCut (↓), and runtime (↓). The corresponding number records as well as analysis of the trade-off between quality and efficiency are given in Appendix C.5.

from the same system (following several underlying distributions), it's promising that one conducts the *offline* training on historical data of one scenario but generalizes the model to other scenarios for fast *online* GP. We consider the challenging application where we train IGP on synthetic graphs but generalize it to real datasets. The two extended applications are elaborated in Appendix D.

5 Conclusion

We proposed an IGP framework to alleviate the NP-hard challenge of GP, obtaining a better trade-off between quality and efficiency. In IGP, we first conduct the *offline* training of a dual GNN structure on historical system snapshots and generalize the model to newly generated graphs for fast high-quality *online* GP. IGP is a generic framework that can incorporate the permutation invariant label information of GP to the *offline* training by combing unsupervised GP objectives (modularity maximization and NCut minimization) with the dual GNN structure. Since we introduced an *inductive* graph embedding scheme and an efficient feature extraction module, IGP can handle the *online* GP with non-fixed number of nodes and clusters. Discussions of possible future work (e.g., scaling IGP up to graphs with large number of nodes and transferring across scenarios) are given in Appendix E.

REFERENCES

Emmanuel Abbe. Community detection and stochastic block models: recent developments. *JMLR*, 18(1):6446–6531, 2017.

Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *JSTAT*, 2008(10):P10008, 2008.

Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. A survey on network embedding. *IEEE TKDE*, 31(5):833–852, 2018.

Lin Dai and Bo Bai. Optimal decomposition for large-scale infrastructure-based wireless networks. *IEEE TWC*, 16(8):4956–4969, 2017.

Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE TPAMI*, 29(11):1944–1957, 2007.

Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *PNAS*, 99(12):7821–7826, 2002.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proc. of ACM SIGKDD*, pp. 855–864, 2016.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proc. of NIPS*, pp. 1024–1034, 2017.

Bruce Hendrickson and Robert W Leland. A multi-level algorithm for partitioning graphs. *Proceedings Supercomputing*, 95(28):1–14, 1995.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. *arXiv preprint arXiv:1809.05343*, 2018.

Brian Karrer and Mark EJ Newman. Stochastic blockmodels and community structure in networks. *Physical Review E*, 83(1):016107, 2011.

Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. Benchmark graphs for testing community detection algorithms. *Physical Review E*, 78(4):046110, 2008.

Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proc. of ACM SIGKDD*, pp. 177–187, 2005.

Jiongqian Liang, Saket Gurukar, and Srinivasan Parthasarathy. Mile: A multi-level framework for scalable graph embedding. *arXiv preprint arXiv:1802.09612*, 2018.

Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv:1511.05644*, 2015.

Azade Nazi, Will Hang, Anna Goldie, Sujith Ravi, and Azalia Mirhoseini. Gap: Generalizable approximate graph partitioning framework. *arXiv:1903.00614*, 2019.

Mark EJ Newman. Modularity and community structure in networks. *PNAS*, 103(23):8577–8582, 2006.

S Pan, R Hu, G Long, J Jiang, L Yao, and C Zhang. Adversarially regularized graph autoencoder for graph embedding. In *Proc. of IJCAI*, 2018.

Siddheshwar V Patil and Dinesh B Kulkarni. Graph partitioning using heuristic kernighan-lin algorithm for parallel computing. In *Next Generation Information Processing System*, pp. 281–288. Springer, 2021.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proc. of ACM SIGKDD*, pp. 701–710, 2014.

Michal Piorkowski, Natasa Sarafijanovic, and Matthias Grossglauser. A parsimonious model of mobile partitioned networks with clustering. In *Proc. of IEEE COMSNETS*, pp. 1–10. IEEE, 2009.

Meng Qin, Kai Lei, Bo Bai, and Gong Zhang. Towards a profiling view for unsupervised traffic classification by exploring the statistic features and link patterns. In *Proc. of SIGCOMM Workshop on Network Meets AI & ML*, pp. 50–56, 2019.

Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. Gcc: Graph contrastive coding for graph neural network pre-training. In *Proc. of ACM SIGKDD*, pp. 1150–1160, 2020.

Ryan Rossi and Nesreen Ahmed. The network data repository with interactive graph analytics and visualization. In *Proc. of AAAI*, pp. 4292–4293, 2015.

Navaratnasothie Selvakkumaran and George Karypis. Multiobjective hypergraph-partitioning algorithms for cut and maximum subdomain-degree minimization. *IEEE TCAD*, 25(3):504–517, 2006.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proc. of WWW*, pp. 1067–1077, 2015.

Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. Verse: Versatile graph embeddings from similarity measures. In *Proc. of WWW*, pp. 539–548, 2018.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics & Computing*, 17(4):395–416, 2007.

Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proc. of ACM SIGKDD*, pp. 1225–1234, 2016.

Fei Wang, Tao Li, Xin Wang, Shenghuo Zhu, and Chris Ding. Community discovery using nonnegative matrix factorization. *DMKD*, 22(3):493–521, 2011.

Klaus Wehrle, Mesut Günes, and James Gross. *Modeling and tools for network simulation*. Springer Science & Business Media, 2010.

Bryan Wilder, Eric Ewing, Bistra Dilkina, and Milind Tambe. End to end learning and optimization on graphs. In *Proc. of NIPS*, pp. 4672–4683, 2019.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proc. of ACM SIGKDD*, pp. 1365–1374, 2015.

Liang Yang, Xiaochun Cao, Dongxiao He, Chuan Wang, Xiao Wang, and Weixiong Zhang. Modularity based community detection with deep learning. In *Proc. of IJCAI*, volume 16, pp. 2252–2258, 2016.

Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, and Sourav S. Bhowmick. Homogeneous network embedding for massive graphs via reweighted personalized pagerank. *VLDB Endow*, 13 (5):670–683, 2020.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graph-saint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

Jie Zhang, Yuxiao Dong, Yan Wang, Jie Tang, and Ming Ding. Prone: Fast and scalable network representation learning. In *Proc. of IJCAI*, volume 19, pp. 4278–4284, 2019.

Ziwei Zhang, Peng Cui, Haoyang Li, Xiao Wang, and Wenwu Zhu. Billion-scale network embedding with iterative random projection. In *Proc. of ICDM*, pp. 787–796. IEEE, 2018.

Jing Zhu, Xingyu Lu, Mark Heimann, and Danai Koutra. Node proximity is all you need: Unified structural and positional node and graph embedding. In *Proc. of SDM*, pp. 163–171. SIAM, 2021.

A PROOF OF THEOREM 1

By the disjoint constraint of GP, i.e., $C_r^t \cap C_s^t = \emptyset$ for $\forall r \neq s$, each node can only belong to one unique cluster. For the *membership indicator matrix* $\mathbf{R}_t$, only one entry in each row of $\mathbf{R}_t$ must be 1 with the rest entries in the same row set to be 0. For the *auxiliary adjacency matrix* $\mathbf{A}_t^{(g)} = \mathbf{R}_t\mathbf{R}_t^T$, consider each entry $(\mathbf{A}_t^{(g)})_{ij} = \sum_{r=1}^{K_t} [(\mathbf{R}_t)_{ir}(\mathbf{R}_t)_{jr}]$ with respect to the node pair (v_i^t, v_j^t).

If v_i^t and v_j^t are in the same cluster C_s^t, i.e., $v_i^t, v_j^t \in C_s^t$, we have $\mathbf{R}_{is} = \mathbf{R}_{js} = 1$. We can further derive $(\mathbf{A}_t^{(g)})_{ij} = (\mathbf{R}_t)_{is}(\mathbf{R}_t)_{js} = 1$. It indicates that there is an edge between (v_i^t, v_j^t) with weight 1, when they are in the same cluster.

If v_i^t and v_j^t are not in the same cluster, i.e., $v_i^t \in C_r^t$ and $v_j^t \in C_s^t$ $(r \neq s)$, we have $\mathbf{R}_{ir} = 1$ and $\mathbf{R}_{js} = 1$, but $(\mathbf{A}_t^{(g)})_{ij} = 0$. Namely, there is no edge between nodes (v_i^t, v_j^t) when they are in different clusters.

In summary, there is an edge between (v_i^t, v_j^t) in the *auxiliary label-induced graph* $\mathcal{G}_t^{(g)}$ only if they are partitioned into the same cluster according to the given partitioning result $C_t = \{C_1^t, \cdots, C_{K_t}^t\}$. Especially, each node $v_i^t \in C_r^t$ (i) must have edges connected to all other nodes in the same cluster C_r^t and (ii) don't have edges connected to nodes in different clusters C_s^t $(r \neq s)$, thus forming a fully connected component with respect to cluster C_r^t. Hence, for a graph $\mathcal{G}_t$ with K_t clusters, its corresponding *auxiliary label-induced graph* $\mathcal{G}_t^{(g)}$ has K_t fully connected components, with each component corresponding to one unique cluster of $\mathcal{G}_t$.

B DETAILED ALGORITHMS OF IGP

In contrast to conventional *transductive* GP approaches applied to a single static graph, IGP is an *inductive* framework that explores the structure characteristics across a set of graphs $S = \{\mathcal{G}_1, \cdots, \mathcal{G}_T\}$. In IGP, we first train a dual GNN structure on historical system snapshots in an *offline* way (regardless of the time cost). We then generalize the trained model to newly generated snapshots for fast high-quality *online* GP without additional optimization, where a better trade-off between quality and efficiency can be achieved. For simplicity, we divide S into a training set Γ_{T}, a validation set Γ_{V}, and a test set Γ', where $\Gamma_{\mathrm{T}} \cup \Gamma_{\mathrm{V}}$ and Γ' represent the set of historical and newly generated system snapshots, respectively. In other words, snapshots in $\Gamma_{\mathrm{T}} \cup \Gamma_{\mathrm{V}}$ should occur before those in Γ'.

B.1 FEATURE EXTRACTION

For a graph snapshot set $S = \{\mathcal{G}_1, \ldots \mathcal{G}_T\}$ with non-fixed node set $\{\mathcal{V}_1, \ldots, \mathcal{V}_T\}$ (non-fixed number of nodes N_t), we introduce an efficient feature extraction strategy via the HEM graph coarsening, which maps the neighbor-induced feature $\mathbf{X}_t \in \Re^{N_t \times N_t}$ ($\mathbf{X}_t = \mathbf{Q}_t$ or $\mathbf{X}_t = \mathbf{M}_t$) to a reduced feature matrix $\mathbf{Z}_t \in \Re^{N_t \times L}$ with fixed dimension L. Algorithm 1 and Algorithm 2 summarize the procedures of HEM graph coarsening and our feature extraction module, respectively.

For each graph $\mathcal{G}_t \in S$, if $L < N_t$, Algorithm 1 first merges $\mathcal{G}_t$ into a supergraph $\mathcal{G}_t^*$ (with L supernodes) based on the reweighted edges $\mathcal{E}_t^{(w)} = \{w(v_i^t, v_j^t) | w(v_i^t, v_j^t) = (\mathbf{X}_t)_{ij}, (\mathbf{A}_t)_{ij} = 1\}$ and outputs a coarsening matrix $\mathbf{C}_t \in \Re^{N_t \times L}$ that describes the merging membership. Concretely, $(\mathbf{C}_t)_{ij} = |v_j^{t*}|^{-0.5}$ if node v_i^t is merged into supernode v_j^{t*} (i.e., $v_i^t \in v_j^{t*}$) and $(\mathbf{C}_t)_{ij} = 0$, otherwise. Based on $\mathbf{C}_t$ given by Algorithm 1, Algorithm 2 further derives the reduced feature matrix $\mathbf{Z}_t$ via $\mathbf{Z}_t = \mathbf{X}_t\mathbf{C}_t$. By the sparsity of $\mathbf{C}_t$, we set its j-th column as $(\mathbf{Z}_t)_{:,j} = \sum_{v_i \in v_j^{t*}} (\mathbf{C}_t)_{ij}(\mathbf{X}_t)_{:,i}$. In contrast, when $L > N_t$, Algorithm 2 directly derives the feature $\mathbf{Z}_t = [\mathbf{X}_t, \mathbf{0}_{N_t \times (L - N_t)}]$ via a simple padding strategy.

Algorithm 1: HEM Graph Coarsening

Input: input graph $\mathcal{G}_t$; neighbor-induced feature $\mathbf{X}_t$; number of nodes N_t; reduced dimension L
Output: coarsening matrix $\mathbf{C}_t$

extract the node set $\mathcal{V}_t^{(0)}$ and reweighted edges $\mathcal{E}_t^{(w)}$ from $\mathcal{G}_t$ and $\mathbf{X}_t$
$\mathcal{E}_t^{(0)} \leftarrow \mathcal{E}_t^{(w)}$ //Initialize the edge set of the 1st coarsening level
$k \leftarrow 0$ //Initialize the level index
$N_t' \leftarrow |\mathcal{V}_t^{(0)}|$ //Initialize the number of (super)nodes
while $N_t' > L$ **do**
  $\mathcal{V}_t^{(k+1)} \leftarrow \emptyset$ //Initialize the node set of next level
  $\mathcal{E}_t^{(k+1)} \leftarrow \emptyset$ //Initialize the edge set of next level
  sort $\mathcal{E}_t^{(k)}$ based on weights in descending order
  arrange the sorted result as a queue $q(\mathcal{E}_t^{(k)})$
  while $q(\mathcal{E}_t^{(k)}) \neq \emptyset$ **do**
    pop $(v_i^{(k)}, v_j^{(k)})$ from $q(\mathcal{E}_t^{(k)})$
    if $v_i^{(k)} \in \mathcal{V}_t^{(k)}$ **and** $v_j^{(k)} \in \mathcal{V}_t^{(k)}$ **then**
      delete $v_i^{(k)}$ and $v_j^{(k)}$ from $\mathcal{V}_t^{(k)}$
      merge $v_i^{(k)}$ and $v_j^{(k)}$ into supernode $v_l^{(k+1)}$
      add $v_l^{(k+1)}$ to $\mathcal{V}_t^{(k+1)}$
      adjust $\mathcal{E}_t^{(k)}$ and $\mathcal{E}_t^{(k+1)}$ w.r.t. $\mathcal{V}_t^{(k)}$ and $\mathcal{V}_t^{(k+1)}$
      $N_t' \leftarrow N_t' - 1$ //Update number of supernodes
    if $N_t' = L$ **then**
      $\mathcal{V}_t^{(k+1)} \leftarrow \mathcal{V}_t^{(k+1)} \cup \mathcal{V}_t^{(k)}$
      return $\mathbf{C}_t$ based on the merging membership of $\mathcal{V}_t^{(k+1)}$
  $\mathcal{V}_t^{(k+1)} \leftarrow \mathcal{V}_t^{(k+1)} \cup \mathcal{V}_t^{(k)}$
  $\mathcal{E}_t^{(k+1)} \leftarrow \mathcal{E}_t^{(k+1)} \cup \mathcal{E}_t^{(k)}$
  $k \leftarrow k + 1$ //Update the level index k
return $\mathbf{C}_t$ based on the merging membership of $\mathcal{V}_t^{(k)}$

Algorithm 2: Feature Extraction of IGP

Input: input graph $\mathcal{G}_t$; neighbor-induced feature $\mathbf{X}_t$; number of nodes N_t; reduced dimension L
Output: reduced feature $\mathbf{Z}_t$

if $L \geq N_t$ **then**
  let $\mathbf{Z}_t = [\mathbf{X}_t, \mathbf{0}_{N_t \times (L - N_t)}]$ //Pad $\mathbf{Z}_t$ with zeros
else if $L < N_t$ **then**
  derive coarsening matrix $\mathbf{C}_t \in \Re^{N_t \times L}$ via Algorithm 1
  derive the merging membership $\mathcal{V}_t^*$ based on $\mathbf{C}_t$
  for each *supernode* $v_j^* \in \mathcal{V}_t^*$ **do**
    $(\mathbf{Z}_t)_{:,j} \leftarrow \mathbf{0}_{N_t \times 1}$//Initialize j-th column of $\mathbf{Z}_t$
    for each *node* $v_i \in v_j^*$ **do**
      $(\mathbf{Z}_t)_{:,j} \leftarrow (\mathbf{Z}_t)_{:,j} + (\mathbf{X}_t)_{:,i}(\mathbf{C}_t)_{ij}$

B.2 Offline Training on Historical System Snapshots

In the *offline* training of IGP, we assume that the partitioning result $C_t = \{C_1^t, \cdots, C_{K_t}^t\}$ of each historical graph $\mathcal{G}_t \in \Gamma_{\mathrm{T}} \cup \Gamma_{\mathrm{V}}$ is available, which is permutation invariant in terms of cluster labels. For simplicity, we correlate each historical graph $\mathcal{G}_t \in \Gamma_{\mathrm{T}} \cup \Gamma_{\mathrm{V}}$ with a 2-tuple $(\mathbf{A}_t, C_t)$. The *offline* training procedure of IGP is summarized in Algorithm 3, where L is the fixed dimension for the reduced feature $\mathbf{Z}_t$; $\{\alpha, \beta\}$ are hyper-parameters of training loss L_G; p is the number of historical snapshots sampled in each epoch; $\{\eta_D, \eta_G\}$ are the learning rates of the optimization algorithm to update $\{\delta_D, \delta_G\}$; n is the maximum number of epochs. For a selected training sample in each epoch, repeatedly updating the model parameters m ($m \geq 1$) times can sometimes lead to better

Algorithm 3: *Offline* Training of IGP (on Historical System Snapshots)

Input: training set Γ_{T}; validation set Γ_{V}; reduced feature dimension L; hyper-parameters of training loss $\{\alpha, \beta\}$; number of historical snapshots sampled in each epoch p; repetitions to update model parameters (w.r.t. each sampled snapshot in a epoch) m; learning rate $\{\eta_D, \eta_G\}$; maximum number of epochs n

Output: best model parameters $\{\delta_G^*, \delta_D^*\}$

initialize the variable $\bar{q}^*$ that saves the best average quality metric
initialize the model parameters $\{\delta_G, \delta_D\}$ via the Xavier algorithm
for $epoch$ **from** 1 **to** n **do**
  for $sample_count$ **from** 1 **to** p **do**
    randomly sample a graph $\mathcal{G}_t \in \Gamma_{\mathrm{T}}$
    get the associated 2-tuple $(\mathbf{A}_t, C_t)$ of $\mathcal{G}_t$
    derive the neighbor-induced feature $\mathbf{X}_t$ based on adjacency matrix $\mathbf{A}_t$
    derive the reduced feature $\mathbf{Z}_t$ from $\{\mathbf{A}_t, \mathbf{X}_t\}$ via Algorithm 2
    derive the membership indicator $\mathbf{H}_t$ based on the partitioning result C_t
    for $update_count$ **from** 1 **to** m **do**
      derive the loss function L_D via equation (8)
      update δ_D via $\delta_D \leftarrow \mathrm{Optimizer}(\eta_D, \delta_D, \frac{\partial L_D}{\partial \delta_D})$ with δ_G fixed
      derive the loss function L_G via equation (12)
      update δ_G via $\delta_G \leftarrow \mathrm{Optimizer}(\eta_G, \delta_G, \frac{\partial L_G}{\partial \delta_G})$ with δ_D fixed
  $\bar{q} \leftarrow 0$ //Initialize the variable that saves the average quality metric of current epoch
  for each $\mathcal{G}_{t'} \in \Gamma_{\mathrm{V}}$ **do**
    get the associated 2-tuple $(\mathbf{A}_{t'}, C_{t'})$ of $\mathcal{G}_{t'}$
    derive the neighbor-induced feature $\mathbf{X}_{t'}$ based on adjacency matrix $\mathbf{A}_{t'}$
    derive the reduced feature $\mathbf{Z}_{t'}$ from $\{\mathbf{A}_{t'}, \mathbf{X}_{t'}\}$ via Algorithm 2
    derive the graph embedding $\mathbf{U}_{t'}$ from generative network G
    apply KMeans to $\mathbf{U}_{t'}$ to derive the partitioning result $\tilde{C}_{t'}$
    evaluate the partitioning quality q based on $\tilde{C}_{t'}$ (and $C_{t'}$)
    $\bar{q} \leftarrow \bar{q} + q$ //Update $\bar{q}$
  $\bar{q} \leftarrow \bar{q}/|\Gamma_V|$ //Compute the average quality metric
  if $\bar{q}$ **better than** $\bar{q}^*$ **then**
    $\{\delta_D^*, \delta_G^*\} \leftarrow \{\delta_D, \delta_G\}$ //Update the best model parameters
    $\bar{q}^* \leftarrow \bar{q}$ //Update the best average partitioning quality

partitioning quality (on the validation and test set). Algorithm 3 finally saves the model parameters $\{\delta_D^*, \delta_G^*\}$ with the best quality metric on validation set Γ_{V}.

B.3 Online Generalization to Newly Generated System Snapshots

In the *online* generalization of IGP, we fix the learned model parameters $\{\delta_D^*, \delta_G^*\}$ and directly derive the graph embedding $\mathbf{U}_{t'}$ with respect to each newly generated graph $\mathcal{G}_{t'} \in \Gamma'$ from the *feature encoder* of G. Since we assume that the number of clusters $K_{t'}$ of each graph $\mathcal{G}_{t'} \in \Gamma'$ is given for the *online* GP, we apply KMeans to $\mathbf{U}_{t'}$ to derive the GP result $C'_{t'}$ with $K_{t'}$ clusters. In this case, the total runtime of the *online* GP for each new graph $\mathcal{G}_{t'} \in \Gamma'$ includes (i) the efficient feature extraction of $\mathbf{Z}_{t'}$, (ii) one feedforward propagation of $\{\mathbf{A}_{t'}, \mathbf{Z}_{t'}\}$ through the *feature encoder*, and (iii) the downstream (KMeans) clustering. In conclusion, we summarize the *online* generalization procedure of IGP in Algorithm 4.

B.4 Complexity Analysis

For a given graph $\mathcal{G}_t$, we use the efficient matrix multiplication to derive the neighbor-induced features $\mathbf{Q}_t$ and $\mathbf{M}_t$. Concretely, we set $\mathbf{Q}_t = \mathbf{A}_t - \mathbf{d}_t^T \mathbf{d}_t / (2w)$ and $\mathbf{M}_t = \mathbf{D}^{-1/2} \mathbf{A}_t \mathbf{D}^{-1/2}$, where $\mathbf{d}_t = [d_1^t, \cdots, d_{N_t}^t]$ and $\mathbf{D}_t = \mathrm{diag}(d_1^t, \cdots, d_{N_t}^t)$ are the degree vector and degree diagonal matrix of $\mathcal{G}_t$. Although the complexities of computing $\mathbf{Q}_t$ and $\mathbf{M}_t$ are both at most $O(N_t^2)$, they can be easily paralleled via GPUs, which significantly reduces the runtime. Moreover, the complexity of the HEM graph coarsening (i.e., Algorithm 1) is no more than $O(N_t |\mathcal{E}_t| \log |\mathcal{E}_t|)$, with $|\mathcal{E}_t|$ denoting

Algorithm 4: *Online* Generalization of IGP (to Newly Generated System Snapshots)

Input: newly generated graph $\mathcal{G}_{t'} \in \Gamma'$, adjacency matrix $\mathbf{A}_{t'}$, reduced feature dimension L, number of clusters $K_{t'}$ (w.r.t. $\mathcal{G}_{t'}$)
Output: partitioning result $\tilde{C}_{t'}$ of $\mathcal{G}_{t'}$

1 derive the neighbor-induced feature $\mathbf{X}_{t'}$ based on adjacency matrix $\mathbf{A}_{t'}$
2 derive the reduced feature $\mathbf{Z}_{t'}$ from $\{\mathbf{A}_{t'}, \mathbf{X}_{t'}\}$ via Algorithm 2
3 derive the graph embedding $\mathbf{U}_{t'}$ from the *feature encoder* of generative network G
4 apply KMeans to $\mathbf{U}_{t'}$ to derive the partitioning result $\tilde{C}_{t'}$ with $K_{t'}$ clusters

the number of edges in $\mathcal{G}_t$ and $|\mathcal{E}_t| \ll N_t^2$. The complexity to derive $\mathbf{Z}_t$ via $\mathbf{X}_t$ and $\mathbf{C}_t$ (i.e., Algorithm 2) is no more than $O(N_t L n^*)$, where n^* is the maximum number of nodes merged into a supernode; L is the number of supernodes (which is also the reduced feature dimension of $\mathbf{Z}_t$). Hence, the complexity to derive the reduced feature matrix $\mathbf{Z}_t$ based on $\mathbf{X}_t$ is no more than $O(N_t|\mathcal{E}_t| \log |\mathcal{E}_t| + N_t L n^*)$.

Let $L^{(l-1)}$ be the feature dimension of the l-th GNN layer of the *feature encoder* or *auxiliary label-induced encoder* in G, where we have $L^{(0)} = L$. We used the efficient sparse-dense matrix multiplication to implement the graph convolutional operation described in equation (5), where the sparsity of $\mathbf{A}_t$ is considered. The complexity of one feedforward propagation through the *feature encoder* is no more than $O(|\mathcal{E}_t|L^{(0)}L^{(1)} + |\mathcal{E}_t|L^{(1)}L^{(2)} + \cdots) = O(|\mathcal{E}_t|LL')$, where $L' = L^{(1)}$ and $L^{(l)} < L^{(l-1)}$. Hence, the overall time complexity of one feedforward propagation through the *feature encoder* is linear in the number of edges of the input graph $\mathcal{G}_t$.

Let M be the dimension of graph embedding. Usually, we have $M \ll N_t$. For a graph with K_t clusters, the complexity of KMeans is no more than $O(N_t K_t M I)$ for I iterations. In particular, the runtime of the downstream clustering module (i.e., the KMeans algorithm) can also be significantly reduced via parallel implementations.

C Experiment Details

C.1 Evaluation Metrics

In our experiments, we used the (unsupervised) metrics of modularity and NCut to evaluate the partitioning quality of all the datasets. The formal definitions of modularity and NCut are given in equation (3) and equation (1) (see Section 2).

For datasets with partitioning ground-truth (i.e., *GN-Net* and *LFR-Net*), we also used the metrics of normalized mutual information (NMI) and accuracy (AC) (Cui et al., 2018), which can measure the correspondence between the partitioning result (given by the method to be evaluated) and ground-truth (given by the dataset). In general, better correspondence indicates better partitioning results. For a graph $\mathcal{G}_t$ with K_t clusters, we use $H_t = \{H_1^t, \cdots, H_{K_t}^t\}$ and $C_t = \{C_1^t, \cdots, C_{K_t}^t\}$ to denote the ground-truth and partitioning result, where $H_r^t \subseteq \mathcal{V}_t$ and $C_r^t \subseteq \mathcal{V}_t$ represent the node member sets of the r-th cluster. Given H_t and C_t, the definition of NMI is as follow:

$$\mathrm{NMI}(H_t, C_t) = \frac{-2\sum_{H_r^t \in H_t}\sum_{C_s^t \in C_t} \frac{n_{r,s}}{n} \log \frac{n \times n_{r,s}}{n_r \times n_s}}{\sum_{H_r^t \in H_t} \frac{n_r}{n} \log \frac{n_r}{n} + \sum_{C_s^t \in C_t} \frac{n_s}{n} \log \frac{n_s}{n}}, \tag{13}$$

where $n_r = |H_r^t|$ is the number of nodes in the r-th cluster of the ground-truth; $n_s = |C_s^t|$ is the number of nodes in the s-th cluster of the partitioning result; $n_{r,s} = |H_r^t \cap C_s^t|$ is the number of nodes that are simultaneously partitioned into the r-th cluster of ground-truth and the s-th cluster of partitioning result. Let $L_t = (l_1^t, \cdots, l_{N_t}^t)$ and $R_t = (r_1^t, \cdots, r_{N_t}^t)$ be the label sequences of the partitioning result and ground-truth with respect to the node sequence $(v_1^t, \cdots, v_{N_t}^t)$, where l_i^t and r_i^t are cluster labels of node v_i^t. The definition of AC is as follow:

$$\mathrm{AC}(R_t, L_t) = \frac{\sum_{i=1}^{N_t} \delta(r_i^t, \mathrm{map}(l_i^t))}{N_t}, \tag{14}$$

where $\mathrm{map}(\cdot)$ is the Kuhn-Munkres mapping function that gives the best membership map from L_t to R_t; $\delta(a, b)$ is the Kronecker delta function with $\delta(a, b) = 1$ when $a = b$ and $\delta(a, b) =$

Table 3: Detailed statistics of the datasets.

	T	Min N	Max N	Avg N	Min $\|\mathcal{E}\|$	Max $\|\mathcal{E}\|$	Avg $\|\mathcal{E}\|$	Min K	Max K	Avg K
GN-0.5	2,000	5,000	5,000	5,000	48,065	49,314	48,752	250	250	250
GN-0.4					48,335	49,698	48,999			
GN-0.3					48,488	49,981	49,246			
L(*f*,0.3)	2,000	5,000	5,000	5,000	22,539	25,707	23,925	57	104	79
L(*f*,0.6)					22,585	25,639	23,927	58	107	79
L(*f*,0.3)	2,000	5,000	5,999	5,503	22,999	29,670	26,311	59	117	87
L(*f*,0.6)		5,000	6,000	5,493	22,680	29,931	26,289	61	118	87
Taxi	3,000	1,279	1,279	1,279	38,554	40,171	39,291	7	12	9
Reddit	1,870	501	3,648	946	525	4,780	1,133	14	121	32
Enron	410	502	3,261	1,048	554	5,097	1,376	10	31	17
AS	733	103	6,474	4,183	487	26,467	16,324	7	39	29

0, otherwise. Usually, larger modularity, NMI, and AC, as well as smaller NCut indicate better partitioning quality of a specific GP method.

C.2 Dataset Details

The detailed statistics of the 11 datasets (with 7 synthetic benchmarks and 4 real datasets) used in our experiments are depicted in Table 3, where T, N, $|\mathcal{E}|$, and K are the number of snapshots, nodes, edges, and clusters, respectively.

GN-Net (Girvan & Newman, 2002) is a widely-used synthetic benchmark for GP, in which the graph topology and partitioning ground-truth of each snapshot can be generated via the stochastic block model (Abbe, 2017). We fixed $N = 5,000$ and evenly partitioned the nodes into $K = 250$ clusters. For two uniformly selected nodes v_i and v_j, if they're set to be in the same cluster, i.e., $v_i, v_j \in C_r$, the probability to generate an edge (v_i, v_j) is p_{in}. Otherwise, i.e., $v_i \in C_r$ and $v_j \in C_s$ ($r \neq s$), the probability to generate an edge (v_i, v_j) is $(1 - p_{\text{in}})/(K - 1)$. We set $p_{in} \in \{0.5, 0.4, 0.3\}$ to generate 3 datasets. For each dataset, we independently generated $2,000$ snapshots. In each snapshot, we also randomly shuffled the node indices to ensure that there's no index correspondence between any two snapshots in the dataset (as we defined in Section 2). Especially, with the decrease of p_{in}, the clustering structures become increasingly difficult to be identified. For simplicity, we denote the *GN-Net* with a specific setting of p_{in} as *GN-*p_{in}. Note that all the datasets of *GN-Net* are with fixed N and K.

LFR-Net (Lancichinetti et al., 2008) is a more challenging synthetic benchmark that can simulate the power-law properties of node degree and cluster size of real-world network systems. It uses $(k, k_{\max}, c_{\min}, c_{\max}, \mu)$ to generate each single graph, where k and $k_{\max}$ are the average and maximum node degree; $c_{\min}$ and $c_{\max}$ are the minimum and maximum cluster size; μ is the mixing ratio between the external degree and total degree of each node v_i with respect to the cluster v_i belongs to. We fixed $(k, k_{\max}, c_{\min}, c_{\max}) = (10, 100, 10, 200)$ and set $\mu \in \{0.3, 0.6\}$. With the increase of μ, the clustering structure is increasingly difficult to be identified. We consider both the cases with fixed and non-fixed N, where we respectively fixed $N = 5,000$ and set N to be a random integer within $[5000, 6000]$ to generate $2,000$ graphs for each case. Similar to *GN-Net*, we also independently generated each snapshot in a dataset and ensure that there's no index correspondence between any two snapshots via the random shuffle of node indices. For simplicity, we use *L*(T, μ) ($T \in \{f, n\}$) to represent the dataset with the specific setting of μ, where f and n denote the case with <u>f</u>ixed and <u>n</u>on-fixed N. Moreover, K is non-fixed in *LFR-Net*.

Taxi[1] (Piorkowski et al., 2009) is a real GPS dataset including the trajectory of $1,279$ taxis in Beijing, which forms a mobile ad hoc network system. We sampled $3,000$ snapshots from the dataset in chronological order. In each snapshot, we treated each taxi as a node and constructed the topology based on the top-50 neighbors with the closest distance for each node in a snapshot. Since *Taxi* doesn't provide ground-truth regarding the number of clusters K for GP, we used the *Louvain* algorithm (Blondel et al., 2008) to determine the proper K value for each snapshot in the dataset. Hence, it's with fixed N and non-fixed K.

Reddit[2] (Yanardag & Vishwanathan, 2015) is a social network dataset extracted from Reddit. Each graph snapshot in the dataset corresponds to an online discussion thread. We extracted $1,870$ graphs

[1] https://www.microsoft.com/en-us/research/publication/t-drive-driving-directions-based-on-taxi-trajectories/

[2] https://github.com/weihua916/powerful-gnns/blob/master/dataset.zip

with $N \geq 500$ from the dataset. Different snapshots are with different number of users (i.e., nodes), varying from 501 to $3,648$. The comment interactions between users in each discussion thread are described as the topology of the corresponding graph. Similarly, since *Reddit* doesn't provide ground-truth regarding the K value, we also used *Louvain* to determine K for each graph. Hence, *Reddit* is with non-fixed N and K. Note that different snapshots have different node sets (i.e., with non-fixed N) and the node indices in each snapshot start from 0, thus there's no correspondence between the node indices of any two snapshots in the dataset.

Enron[3] (Rossi & Ahmed, 2015) is a real interaction network extracted from the email system of the Enron company, which contains the email records from 1980-01-01 to 2004-02-04. We extracted 410 snapshots with $N \geq 500$ from the records in chronological order, in which we treated each user as a unique node and constructed the topology based on the email interactions. Note that different snapshots have different sets of users that have interaction behaviors. Moreover, we also used the *Louvain* algorithm to estimate the value of K for each snapshot, since the dataset doesn't provide the ground-truth regarding K. Therefore, it's with non-fixed N and K. We also ensure that there's no node index correspondence between any two graphs in the dataset by permuting the node indices of each graph from 0.

AS[4] (Leskovec et al., 2005) is a real dataset extracted from the border gateway protocol (BGP) logs of a communication network, which describe the topology among a set of autonomous systems (AS). It contains 733 daily instances spanning from 1997-08-11 to 2000-01-02. In the dataset, each BGP router is abstracted as a unique node with the communication between routers extracted as the graph topology. As the dataset allows the addition and deletion of routers, different snapshots are with different set of nodes. Since *AS* doesn't provide ground-truth regarding the K value, we also applied *Louvain* to determine K for each snapshot. Therefore, it's with non-fixed N and K, where there's no index correspondence between any two snapshots.

In summary, the 3 datasets of *GN-Net* are with fixed N and K. The 4 datasets of *LFR-Net* cover both the cases of (i) fixed N but non-fixed K as well as (ii) non-fixed N and K. *Taxi* is a real dataset with fixed N and non-fixed K, while *Reddit*, *Enron*, and *AS* are with non-fixed N and K. Note that both *GN-Net* and *LFR-Net* are synthetic benchmarks that can simultaneously generate the graph topology and partitioning ground-truth of each snapshot. *Taxi*, *Reddit*, *Enron*, and *AS* are real datasets without partitioning ground-truth (regarding the value of K), so we applied the *Louvain* algorithm to these datasets to determine the proper K value of each single snapshot, which can ensure the fairness of comparison, i.e., each snapshot is with a common K for all the GP method to be evaluated.

C.3 Baseline Methods

In addition to the 15 baseline methods introduced in Section 4, we also evaluated the partitioning quality and runtime of other 5 strong baselines, which include degree-corrected stochastic block model (*DCSBM*) (Karrer & Newman, 2011), *DeepWalk* (Perozzi et al., 2014), *node2vec* (Grover & Leskovec, 2016), *LINE* (Tang et al., 2015), and *GIN* (Xu et al., 2018). In summary, we have in total evaluated the quality and efficiency of 20 baselines with 4 categories.

- First, *SNMF*, *DCSBM*, and *SC* are classic *transductive* GP methods based on several GP objectives (e.g., *SC* for NCut minimization and *DCSBM* for modularity maximization), which aim to achieve high-quality partitioning results.
- Second, *DeepWalk*, *node2vec*, and *LINE*, *VERSE*, *MILE*, *PhUSION*, *NPR*, *RandNE*, and *ProNE* are unsupervised *transductive* graph embedding baselines. In particular, *LINE*, *NPR*, *RandNE*, and *ProNE* are efficient embedding methods with several fast approximation strategies for the embedding learning. In contrast, *DeepWalk*, *node2vec*, *VERSE*, *MILE*, and *PhUSION* are baselines that aim to derive high-quality representations.
- Third, *Metis*, *hMetis*, and *GraClus* are classic fast *transductive* GP baselines that aim to achieve low runtime with heuristic approximation of classic GP objectives, which may suffer from quality degradation due to the information loss of heuristic approximation.
- Fourth, *GraSAGE*, *GAT*, and *GIN*, *GAP*, and *ClusNet* are *inductive* GNN baselines. In particular, *GraSAGE*, *GAT*, and *GIN* are SOTA *inductive* GNN following the graph embed-

[3] http://networkrepository.com/ia-enron-email-dynamic.php

[4] http://snap.stanford.edu/data/as-733.html

ding scheme, i.e., learning node representations to support downstream tasks (including GP) based on unsupervised loss. Note that we don't consider the supervised training loss of these *inductive* GNNs (e.g., using the cross-entropy loss for a softmax output layer combined with the GNN), because GP is a typical unsupervised task where the cluster labels are permutation invariant. In contrast, *GAP* and *ClusNet* are E2E GP baselines based on the *inductive* nature of GNN, in which an additional FC output layer is integrated in the GNN to directly output the partitioning results.

Note that the first three types of methods are inherently *transductive*, which only focus on the GP on a single static graph but cannot be generalized to other new graphs. Hence, we had to apply these *transductive* baselines to each newly generated system snapshot (i.e., each snapshot in the test set Γ' of a dataset) from scratch. For the fourth type of baselines (i.e., *inductive* GNN approaches), we used the same settings with IGP (i.e., *offline* training on the training set Γ_{T} and *online* generalization to the test set Γ'). In particular, we can only utilize the *inductive* nature of the E2E GNN methods (i.e., *GAP* and *ClusNet*) on the datasets with fixed number of clusters K (i.e., on *GN-Net*). However, we have to use the same settings with *transductive* baselines (i.e., training *GAP* and *ClusNet* from scratch for each new test graph) on the rest datasets with non-fixed K, due to the fixed dimension of their output layers. Some other settings regarding the baseline methods have been already introduced in Section 4.

C.4 Implementation Details

In our experiments, we used PyTorch to implement the dual GNN structure of IGP. The proposed feature extraction module (i.e., Algorithm 1 and Algorithm 2) was implemented via NumPy and PyTorch, while we used the implementation of KMeans of scikit-learn as the downstream clustering module. All the experiments were conducted on a server with Intel Xeon CPU (E5-2650v4@2.20GHz, 48 cores), 1 Tesla V100 GPU, 61GB memory, as well as the Ubuntu Linux operating system. In this setting, modules implemented by PyTorch were speeded up via the GPU.

For all the embedding-based methods including the *transductive* embedding baselines (i.e., the second type of baseline), *inductive* GNNs (i.e., *GraSAGE*, *GAT*, and *GIN*), and IGP, we used KMeans as the downstream clustering module. In contrast, we can directly obtain the partitioning results from the outputs of the rest baselines. Note that the KMeans clustering algorithm for graph-embedding-based methods and optimization algorithms of several classic GP baselines (i.e., *SNMF*, *DCSBM*, *Metis*, *hMetis*, and *GraClus*) have the limitation of obtaining local optimal solutions. We adopted the strategy to run the KMeans of embedding-based methods and optimization algorithms of several classic GP baselines 10 times, which was included in the total runtime of the corresponding methods. The partitioning membership with the minimum loss function value (e.g., the sum of squared distance between data and cluster center for KMeans) was reported as the final partitioning result for evaluation. When testing the runtime of each method, we ensured that only the method to be evaluated was running on the experiment server, i.e., there were no multiple methods simultaneously running on the test server.

On the validation set Γ_{V} of each dataset, we tuned the parameters (i.e., $\{\alpha, \beta, m\}$) and determine the proper layer configuration for the two variants of IGP. Concretely, we adjusted $\alpha \in \{1, 10\}$ and $m \in \{1, 2, \cdots 5\}$ for both IGP-M and IGP-C, while we tuned $\beta \in \{0.1, 1, 2, \cdots 5\}$ and $\beta \in \{0.1, 1, 100, 1000\}$ for IGP-M and IGP-C, respectively. The recommended parameter settings and layer configurations of IGP-M and IGP-C with respect to each dataset are depicted in Table 4, where dimensions of the first and last layer of the generative network G are set to be L (i.e., the dimension of reduced feature) and M (i.e., the dimension of graph embedding), respectively. Note that we use the same layer configuration for IGP-M and IGP-C on each dataset. Furthermore, we set the maximum number of epochs $n = 100$ for all the datasets.

C.5 Detailed Evaluation Results and Discussions

C.5.1 Quantitative Evaluation

In the experiments, we recorded the mean value μ and standard deviation σ of all the evaluation metrics and runtime for all the methods (on the test set Γ' of each dataset). The quantitative evaluation results on *GN-Net* and *LFR-Net* in terms of NMI and AC are depicted in Table 5 and Table 6,

Table 4: Parameter settings and layer configurations of IGP.

	IGP-M		**IGP-C**		Layer Configurations	
	(α, β, m, p)	(η_G, η_D)	(α, β, m, p)	(η_G, η_D)	G	D
GN-0.5	(1,1,1,1000)	(5e-4,5e-4)	(1,1,1,1000)	(1e-4,1e-4)	2048-1024-512	512-128-64-16-1
GN-0.4	(1,1,1,1000)	(5e-4,5e-4)	(1,1,1,1000)	(1e-4,1e-4)		
GN-0.3	(1,5,1,1000)	(5e-4,5e-4)	(1,1,1,1000)	(1e-4,1e-4)		
L(*f*,0.3)	(1,1,1,1000)	(5e-4,5e-4)	(1,1e2,5,1000)	(1e-4,1e-4)	4096-2048-512-256	256-128-64-16-1
L(*f*,0.6)	(1,3,1,1000)	(1e-4,1e-4)	(1,1e3,2,1000)	(5e-5,5e-5)		
L(*n*,0.3)	(1,1,1,1000)	(5e-4,5e-4)	(1,1e2,5,1000)	(1e-4,1e-4)	4096-2048-512-256	256-128-64-16-1
L(*n*,0.6)	(1,3,1,1000)	(1e-4,1e-4)	(1,1e3,2,1000)	(5e-5,5e-5)		
Taxi	(1,0.1,1,2000)	(5e-5,5e-5)	(1,1e2,1,2000)	(5e-5,5e-5)	1024-512-256	256-64-32-16-1
Reddit	(1,0.1,1,1000)	(5e-5,5e-5)	(1,0.1,1,1000)	(5e-4,5e-4)	2048-1024-512-256	256-64-16-1
AS	(1,0.1,1,500)	(5e-5,5e-5)	(1,1e2,1,500)	(5e-5,5e-5)	6000-4096-2048-1024-256	256-128-64-16-1
Enron	(1,5,1,300)	(5e-5,5e-5)	(1,1e2,1,300)	(5e-5,5e-5)	1024-512-256	256-128-64-16-1

Table 5: Quantitative evaluation results in terms of NMI (%) (↑).

	GN-0.5	*GN*-0.4	*GN*-0.3	*L*(*f*,0.3)	*L*(*f*,0.6)	*L*(*n*,0.3)	*L*(*n*,0.6)
SNMF	99.53 (0.21)	98.79 (0.36)	94.37 (0.69)	83.53 (1.60)	49.15 (2.56)	83.59 (1.59)	49.98 (2.19)
DCSBM	67.92 (10.49)	57.68 (10.42)	49.76 (6.04)	45.64 (6.25)	27.27 (2.89)	44.73 (7.48)	27.32 (2.42)
SC	99.75 (0.10)	98.73 (0.24)	94.26 (0.41)	98.19 (0.39)	35.78 (3.65)	98.21 (0.35)	24.66 (2.76)
DeepWalk	99.40 (0.11)	97.60 (0.30)	82.81 (0.85)	93.27 (0.77)	55.71 (2.19)	93.36 (0.78)	56.40 (2.03)
node2vec	99.44 (0.09)	98.89 (0.19)	88.26 (0.77)	95.15 (0.73)	59.11 (2.21)	95.22 (0.68)	59.88 (2.07)
LINE	93.73 (0.54)	68.70 (0.75)	53.25 (0.31)	80.72 (1.44)	42.69 (2.59)	79.89 (1.56)	38.78 (3.29)
VERSE	92.09 (0.99)	84.75 (1.57)	61.15 (1.50)	89.80 (0.89)	44.97 (3.16)	89.94 (0.97)	45.87 (3.05)
MILE	99.79 (0.04)	99.17 (0.13)	93.39 (0.48)	93.87 (0.95)	20.37 (2.32)	94.11 (0.97)	20.67 (2.04)
PhUSION	99.46 (0.08)	98.81 (0.20)	93.89 (0.57)	94.85 (0.75)	54.18 (2.64)	94.90 (0.72)	55.52 (2.38)
NRP	94.98 (0.78)	90.73 (0.86)	77.90 (1.24)	55.78 (3.02)	26.60 (2.39)	51.90 (3.05)	22.26 (2.52)
RandNE	98.33 (0.26)	94.08 (0.47)	79.18 (1.01)	57.63 (2.06)	18.57 (2.16)	58.08 (2.08)	18.87 (1.95)
ProNE	99.80 (0.05)	99.54 (0.08)	96.62 (0.36)	93.33 (0.81)	55.88 (3.04)	93.31 (0.89)	57.61 (2.63)
Metis	98.68 (0.33)	94.68 (0.41)	80.86 (0.77)	84.94 (0.74)	44.52 (2.48)	85.43 (0.71)	45.53 (2.28)
hMetis	96.30 (0.16)	93.82 (0.30)	82.08 (0.68)	88.85 (1.27)	41.65 (3.28)	89.13 (1.17)	42.64 (2.99)
GraClus	99.54 (0.19)	98.10 (0.30)	90.72 (0.76)	90.58 (1.14)	53.06 (3.00)	90.60 (1.09)	53.90 (2.75)
GraSAGE	45.76 (0.13)	45.98 (0.09)	43.21 (0.21)	22.52 (2.21)	15.79 (1.89)	24.67 (2.03)	16.66 (1.77)
GAT	46.09 (0.24)	45.03 (0.25)	44.55 (0.20)	22.99 (1.88)	15.49 (1.78)	25.91 (1.94)	15.83 (1.70)
GIN	43.29 (0.19)	43.26 (0.19)	43.49 (0.17)	18.49 (1.83)	13.32 (1.47)	17.61 (1.71)	13.43 (1.22)
GAP	72.69 (0.52)	64.37 (0.39)	56.13 (0.31)	93.73 (1.76)	47.30 (3.23)	93.64 (1.65)	48.01 (3.11)
ClusNet	26.30 (1.85)	17.65 (2.10)	13.10 (1.44)	89.84 (1.31)	55.49 (2.04)	90.10 (1.27)	56.04 (1.78)
IGP-M	99.85 (0.05)	99.38 (0.11)	96.26 (0.32)	97.21 (0.49)	56.07 (2.33)	97.30 (0.44)	56.48 (2.08)
IGP-C	99.85 (0.04)	99.46 (0.09)	95.50 (0.43)	96.44 (0.57)	52.11 (2.79)	96.56 (0.59)	52.71 (2.52)

respectively. The evaluation results on all the datasets in terms of modularity and NCut are illustrated in Table 7, Table 8, Table 9, and Table 10. Moreover, Table 11 and Table 12 give the runtime of all the methods. For both IGP-M and IGP-C, we also recorded the time of (i) feature extraction (i.e., Algorithm 2), (ii) one feedforward propagation (through the *feature encoder* of G), and (iii) downstream clustering (i.e., 10 independent runs of KMeans), which together contribute to the total runtime of IGP. The runtime of the 3 modules is denoted as 'Feat', 'Prop', and 'Clus', respectively. In particular, we use the format of 'μ (σ)' to represent the evaluation result.

According to the evaluation results in Table 5, Table 6, Table 7, Table 8, Table 9, Table 10 Table 11, and Table 12, which are consistent with the visualization results in Fig. 3, we have the following major observations.

- For all the datasets, IGP-M and IGP-C are in the top group with the best quality metrics. Moreover, their runtime is also competitive to the fast GP baselines (e.g., *Metis*, *hMetis*, *GraClus*, *RandNE*, etc.) and other *inductive* GNNs with fast *online* generalization (e.g., *GraSAGE*, *GAT* and *GIN*). From the perspective of handling the NP-hard challenge of GP, IGP achieves a significant trade-off between the quality and efficiency of *online* GP on newly generated system snapshots.

- Surprisingly, in some cases, IGP even has the best or second-best quality metrics. Note that IGP is a novel *inductive* framework, in which we don't conduct additional optimization on the test set of each dataset, but other conventional *transductive* baselines (e.g., *SP*, *DeepWalk*, etc.) can fully explore the structural properties of each test graph. It indicates the potential of IGP to capture additional characteristics shared by multiple historical snapshots beyond conventional *transductive* approaches applied to one single new graph.

- On *GN-Net* and *LFR-Net*, the partitioning quality of all the methods declines with the decrease of p_{in} and increase of μ, where both IGP-M and IGP-C are always in the top group with best quality metrics. It demonstrates the strong robustness of IGP, especially when the intrinsic clustering structure of the given graph isn't distinct.

Table 6: Quantitative evaluation results in terms of AC (%) (↑).

	GN-0.5	*GN*-0.4	*GN*-0.3	*L*(*f*,0.3)	*L*(*f*,0.6)	*L*(*n*,0.3)	*L*(*n*,0.6)
SNMF	98.70 (0.50)	97.46 (0.78)	90.25 (1.32)	73.37 (3.20)	41.61 (2.56)	72.96 (2.97)	41.95 (2.40)
DCSBM	38.08 (13.89)	25.83 (9.88)	16.40 (3.93)	22.35 (4.39)	13.65 (2.02)	21.37 (4.96)	13.04 (1.75)
SC	98.69 (0.39)	96.67 (0.54)	89.29 (0.85)	95.09 (0.73)	26.01 (3.62)	97.27 (0.96)	12.26 (1.59)
DeepWalk	96.40 (0.58)	93.15 (0.91)	69.03 (1.52)	87.91 (2.01)	54.97 (2.10)	87.65 (2.13)	55.12 (1.97)
node2vec	96.48 (0.51)	95.53 (0.64)	79.30 (1.47)	90.01 (2.10)	58.40 (2.07)	89.96 (2.05)	58.79 (1.96)
LINE	86.17 (1.06)	46.90 (1.34)	20.36 (0.66)	72.14 (3.07)	38.22 (2.24)	70.55 (3.17)	33.63 (3.50)
VERSE	80.76 (1.89)	70.02 (2.53)	35.39 (2.16)	83.89 (2.07)	40.25 (3.33)	83.56 (2.14)	41.03 (3.11)
MILE	98.67 (0.31)	96.91 (0.50)	87.48 (0.91)	92.60 (1.60)	10.19 (0.93)	92.57 (1.55)	9.94 (0.86)
PhUSION	96.49 (0.54)	94.76 (0.66)	86.77 (1.11)	89.14 (2.06)	51.83 (2.49)	88.80 (1.75)	52.75 (2.25)
NRP	81.61 (1.36)	72.90 (1.26)	56.39 (1.78)	42.27 (2.46)	20.42 (1.74)	38.75 (2.64)	16.43 (1.70)
RandNE	95.07 (0.68)	86.56 (1.03)	63.77 (1.49)	40.67 (1.77)	12.91 (1.25)	40.64 (1.73)	12.80 (1.17)
ProNE	98.69 (0.36)	97.66 (0.42)	92.16 (0.80)	82.56 (2.55)	52.04 (2.95)	82.23 (2.52)	53.51 (2.76)
Metis	97.90 (0.62)	91.25 (0.82)	66.48 (1.55)	58.70 (2.28)	35.13 (1.77)	58.58 (2.57)	35.37 (1.55)
hMetis	88.01 (0.44)	84.44 (0.72)	66.84 (1.20)	66.62 (3.16)	32.80 (2.91)	65.99 (2.49)	33.26 (2.63)
GraClus	97.54 (0.84)	95.11 (0.94)	83.16 (1.54)	72.00 (3.40)	46.19 (3.02)	71.81 (3.33)	46.50 (2.78)
GraSAGE	7.83 (0.13)	7.66 (0.13)	7.44 (0.12)	7.55 (0.25)	5.79 (0.15)	7.58 (0.34)	5.66 (0.19)
GAT	9.20 (0.17)	8.66 (0.16)	8.00 (0.13)	8.52 (0.36)	6.22 (0.16)	8.78 (0.40)	5.69 (0.19)
GIN	7.61 (0.12)	7.51 (0.11)	7.46 (0.11)	6.97 (0.23)	5.83 (0.15)	6.08 (0.30)	5.37 (0.21)
GAP	38.90 (1.04)	29.71 (0.76)	20.30 (0.57)	83.63 (4.16)	40.06 (3.34)	83.06 (3.92)	40.08 (3.41)
ClusNet	4.51 (0.30)	2.94 (0.28)	3.13 (0.23)	86.45 (2.47)	56.20 (1.82)	86.87 (2.36)	56.25 (1.65)
IGP-M	98.96 (0.28)	97.36 (0.46)	91.96 (0.73)	94.29 (1.41)	53.57 (2.21)	94.24 (1.38)	53.42 (2.04)
IGP-C	98.93 (0.31)	97.48 (0.46)	90.47 (0.85)	92.58 (1.49)	47.38 (2.68)	92.46 (1.46)	47.47 (2.59)

Table 7: Quantitative evaluation results in terms of Modularity (%) (↑) on datasets with ground-truth.

	GN-0.5	*GN*-0.4	*GN*-0.3	*L*(*f*,0.3)	*L*(*f*,0.6)	*L*(*n*,0.3)	*L*(*n*,0.6)
SNMF	47.95 (0.23)	37.79 (0.27)	27.28 (0.29)	56.12 (1.58)	29.22 (0.63)	56.03 (1.50)	29.43 (0.60)
DCSBM	17.31 (5.55)	9.33 (2.40)	3.86 (0.72)	11.37 (2.46)	4.77 (0.96)	11.00 (2.68)	4.71 (0.74)
SC	48.13 (0.20)	37.81 (0.23)	27.07 (0.23)	67.32 (0.20)	26.64 (1.06)	67.50 (0.26)	23.62 (2.83)
DeepWalk	47.88 (0.20)	37.27 (0.28)	21.97 (0.46)	62.31 (1.11)	33.92 (0.54)	62.35 (1.14)	34.06 (0.48)
node2vec	47.89 (0.19)	37.91 (0.21)	24.62 (0.45)	63.23 (1.14)	34.47 (0.54)	63.30 (1.14)	34.64 (0.47)
LINE	42.47 (0.59)	15.96 (0.61)	5.14 (0.17)	55.88 (1.84)	23.34 (0.98)	55.21 (1.74)	20.76 (1.93)
VERSE	41.33 (0.91)	28.62 (1.05)	12.79 (0.69)	60.50 (1.18)	29.34 (1.13)	60.52 (1.19)	29.40 (1.08)
MILE	48.21 (0.18)	38.08 (0.20)	26.96 (0.27)	66.37 (0.34)	21.33 (1.25)	66.55 (0.42)	21.59 (1.14)
PhUSION	47.89 (0.20)	37.79 (0.22)	27.03 (0.29)	63.25 (1.00)	33.80 (0.50)	63.26 (0.91)	34.06 (0.48)
NRP	45.21 (0.41)	34.31 (0.39)	22.27 (0.46)	24.32 (1.29)	7.07 (0.72)	25.00 (1.79)	5.69 (0.80)
RandNE	47.29 (0.24)	35.61 (0.34)	21.08 (0.47)	18.44 (1.20)	3.26 (0.34)	18.48 (1.21)	3.38 (0.32)
ProNE	48.20 (0.18)	38.20 (0.19)	27.92 (0.24)	60.96 (1.19)	32.64 (0.81)	60.98 (1.17)	33.24 (0.70)
Metis	47.29 (0.34)	35.24 (0.30)	21.62 (0.36)	52.38 (1.12)	30.51 (0.44)	52.41 (1.17)	30.58 (0.39)
hMetis	44.78 (0.22)	34.62 (0.28)	22.39 (0.31)	57.51 (1.14)	31.80 (0.60)	57.44 (0.96)	31.99 (0.49)
GraClus	47.83 (0.26)	37.34 (0.30)	25.00 (0.48)	53.54 (1.98)	31.61 (0.92)	53.43 (1.87)	31.71 (0.83)
GraSAGE	0.08 (0.03)	0.04 (0.03)	0.01 (0.01)	0.28 (0.17)	0.18 (0.11)	0.56 (0.27)	0.17 (0.16)
GAT	0.48 (0.05)	0.27 (0.05)	0.08 (0.03)	0.99 (0.17)	0.08 (0.12)	1.71 (0.22)	0.02 (0.11)
GIN	0.05 (0.03)	0.03 (0.03)	0.01 (0.03)	0.50 (0.11)	0.06 (0.08)	4.73 (0.29)	4.19 (0.18)
GAP	19.35 (0.63)	9.95 (0.36)	5.24 (0.19)	66.45 (0.59)	32.96 (0.73)	66.65 (0.57)	32.96 (0.77)
ClusNet	0.03 (0.10)	0.02 (0.12)	0.01 (0.13)	65.52 (0.53)	35.10 (0.51)	65.77 (0.54)	35.25 (0.44)
IGP-M	48.25 (0.18)	38.17 (00.20)	27.92 (0.22)	66.26 (0.54)	34.14 (0.53)	66.45 (0.56)	34.22 (0.47)
IGP-C	48.25 (0.18)	38.19 (00.19)	27.68 (0.25)	65.72 (0.59)	31.81 (0.65)	65.83 (0.64)	31.95 (0.62)

- Similar to IGP, the E2E GNN baselines (i.e., *GAP* and *ClusNet*) can also apply the fast *online* generalization to datasets with fixed K (i.e., on *GN-Net*). Namely, we also trained *GAP* and *ClusNet* on the training set of *GN-Net* and directly generalized them to the corresponding test sets for *online* GP, in which we directly derived the partitioning results from their output layers. However, the *online* GP of *GAP* and *ClusNet* is with poor quality. It indicates that the *online* generalization of the graph embedding scheme in IGP is more robust than that of the E2E approaches, especially when clustering structure of a dataset isn't distinct. Furthermore, we still need to train *GAP* and *ClusNet* from scratch for each new test graph on datasets with non-fixed K due to the fixed dimensions of their output layers, which is time-consuming. In contrast, the graph embedding scheme of IGP can still handle the fast *online* GP with non-fixed K.
- In contrast to the reduced structural feature $\mathbf{Z}_t$ in IGP, we used a constant matrix $\mathbf{1}_{N_t \times L}$ as the feature inputs of inductive GNN baselines (i.e., *GraSAGE*, *GAT*, adn *GIN*), which is a standard setting of most *inductive* GNNs for graphs without available node attributes (Xu et al., 2018). The widely-used unsupervised training loss of *GraSAGE* was also adopted in the *offline* training of these inductive GNNs. Although the inductive GNN baselines have slightly faster runtime over IGP, they still suffer from poor partitioning quality on all the datasets. The results further validate that the extracted feature $\mathbf{Z}_t$ as well as the hybrid training loss of IGP (that can incorporate the permutation invariant label information of historical training snapshots) are essential to ensure its high partitioning quality.
- For the total runtime of IGP on all the datasets, the downstream clustering (e.g., KMeans with 10 independent runs) is the major bottleneck. However, the downstream clustering

Table 8: Quantitative evaluation results in terms of Modularity (%) (↑) on datasets w/o ground-truth.

	Taxi	*Reddit*	*AS*	*Enron*
SNMF	73.72 (0.41)	68.29 (7.16)	53.32 (2.55)	73.88 (14.55)
DCSBM	73.71 (0.60)	2.81 (5.17)	4.97 (2.16)	4.36 (7.09)
SC	73.80 (0.59)	61.19 (7.68)	43.14 (6.60)	71.43 (19.05)
DeepWalk	73.72 (0.54)	65.66 (8.91)	56.48 (2.90)	74.72 (17.57)
node2vec	73.64 (0.54)	69.31 (7.90)	58.19 (2.80)	76.86 (13.31)
LINE	73.95 (0.54)	57.25 (12.92)	45.02 (4.87)	70.94 (14.86)
VERSE	73.69 (0.51)	51.70 (13.77)	52.25 (10.59)	73.43 (18.95)
MILE	72.91 (1.21)	59.09 (17.94)	56.83 (3.20)	74.78 (18.98)
PhUSION	72.93 (0.59)	67.69 (6.18)	58.18 (2.78)	67.84 (12.37)
NRP	6.07 (9.55)	4.01 (5.99)	4.61 (3.39)	0.24 (6.53)
RandNE	73.12 (0.72)	5.52 (5.04)	5.30 (0.54)	21.53 (12.32)
ProNE	73.61 (0.64)	62.88 (5.27)	49.10 (3.87)	62.80 (9.98)
Metis	70.48 (0.51)	52.34 (10.68)	52.75 (3.86)	65.04 (17.07)
hMetis	73.44 (0.80)	54.27 (11.02)	56.03 (4.33)	67.54 (18.16)
GraClus	73.05 (0.64)	45.27 (11.59)	46.81 (4.83)	62.02 (21.93)
GraSAGE	21.85 (1.11)	6.94 (3.72)	5.69 (1.47)	9.41 (6.55)
GAT	1.11 (0.42)	8.69 (11.82)	4.60 (0.84)	4.20 (4.89)
GIN	8.09 (0.44)	4.82 (7.30)	5.62 (7.60)	10.82 (3.61)
GAP	71.79 (1.30)	65.08 (6.24)	58.31 (2.68)	75.47 (12.57)
ClusNet	72.54 (1.27)	55.31 (10.37)	55.87 (2.42)	72.56 (18.75)
IGP-M	74.01 (0.51)	69.38 (5.49)	56.82 (2.71)	74.33 (13.51)
IGP-C	74.00 (0.42)	69.37 (5.47)	55.40 (2.26)	74.15 (11.78)

Table 9: Quantitative evaluation results in terms of NCut (↓) on datasets with ground-truth.

	GN-0.5	*GN*-0.4	*GN*-0.3	*L*(*f*,0.3)	*L*(*f*,0.6)	*L*(*n*,0.3)	*L*(*n*,0.6)
SNMF	139.1402 (10.8984)	213.2316 (14.2808)	352.0424 (8.9401)	38.7998 (5.6174)	98.4915 (10.9462)	43.2193 (6.3339)	106.7235 (11.8945)
DCSBM	3.74e3 (2.33e3)	7.64e3 (4.37e3)	1.73e4 (6.21e3)	3.26e3 (1.72e3)	2.25e3 (1.63e3)	4.10e3 (2.14e3)	2.51e3 (1.70e3)
SC	294.6745 (388.3055)	286.0447 (118.3960)	342.3563 (34.2894)	35.3769 (35.3392)	115.4836 (29.3025)	40.2047 (45.3507)	88.1085 (43.1658)
DeepWalk	283.4350 (114.4880)	488.6243 (202.1361)	1.05e3 (298.5933)	36.2941 (39.0004)	80.1671 (16.2045)	42.3863 (50.8756)	89.7615 (35.8299)
node2vec	224.8467 (84.6247)	346.0660 (142.8872)	664.8052 (193.2850)	55.1128 (94.3308)	80.5008 (37.9300)	50.5657 (63.5605)	87.0301 (34.3899)
LINE	374.6509 (251.9787)	1.25e3 (331.8251)	3.04e3 (532.2893)	2.56e3 (1.21e3)	2.20e3 (1.30e3)	2.83e3 (1.33e3)	1.93e3 (1.12e3)
VERSE	363.7127 (97.6181)	463.8922 (64.8920)	939.2451 (56.1713)	43.0730 (44.9574)	94.5368 (11.1583)	47.9853 (70.4266)	1.04e2 (11.0544)
MILE	241.9054 (86.4033)	262.4250 (81.6662)	356.6205 (7.2723)	36.8684 (62.4733)	60.6043 (7.5151)	33.3624 (32.6776)	65.9406 (9.74670)
PhUSION	160.1741 (36.1058)	222.9295 (5.2864)	360.9837 (7.6685)	24.8669 (4.0223)	74.2453 (7.6983)	27.2123 (3.5929)	80.5383 (8.4627)
NRP	806.64155 (190.8790)	2.18e3 (374.9016)	5.82e3 (621.3819)	1.91e3 (660.6990)	7.48e3 (1.06e3)	3.84e3 (848.7790)	9.63e3 (1.36e3)
RandNE	464.8355 (199.6189)	1.36e3 (372.7496)	5.07e3 (877.5772)	2.24e3 (1.50e3)	1.31e4 (2.30e3)	2.83e3 (1.83e3)	1.41e4 (2.56e3)
ProNE	224.3518 (77.4681)	295.3720 (92.0792)	696.4169 (314.1032)	47.0308 (125.6592)	78.7813 (8.5111)	59.3407 (168.7844)	85.0403 (9.6514)
Metis	141.1405 (3.8763)	242.9069 (6.5644)	502.6296 (12.1616)	38.5993 (5.7621)	87.2077 (9.5311)	42.8839 (7.7193)	94.9605 (10.4598)
hMetis	189.0319 (4.8480)	304.8727 (11.9203)	531.0491 (13.9798)	69.7714 (97.0502)	76.7831 (7.4399)	80.5635 (129.0918)	83.3548 (8.7206)
GraClus	150.2571 (37.6267)	229.8709 (74.7051)	396.5405 (24.4437)	39.5468 (7.9081)	79.5679 (9.2519)	44.0415 (7.7275)	86.7427 (10.0247)
GraSAGE	1.87e5 (1.23e4)	2.05e5 (1.29e4)	1.42e5 (1.17e4)	3.42e4 (4.58e3)	3.71e4 (5.51e3)	4.79e4 (6.36e3)	5.50e4 (6.79e3)
GAT	7.70e4 (9.07e3)	9.49e4 (9.97e3)	1.36e5 (1.27e4)	1.81e4 (5.16e3)	3.17e4 (5.66e3)	3.26e4 (5.31e3)	3.40e4 (6.86e3)
GIN	1.36e5 (1.054e4)	1.45e5 (1.10e4)	1.53e5 (1.11e4)	2.42e4 (6.691e3)	2.77e4 (7.50e3)	4.45e4 (8.52e3)	4.05e4 (9.36e3)
GAP	4.38e3 (1.21e3)	7.99e3 (2.00e3)	1.28e4 (2.70e3)	17.3294 (2.7376)	69.5463 (7.8034)	18.3470 (2.5146)	76.2432 (9.5541)
ClusNet	5.58e4 (8.38e3)	2.80e4 (6.12e3)	2.91e4 (5.12e3)	2.97e2 (1.66e2)	1.11e3 (4.18e2)	2.91e2 (1.56e2)	1.19e3 (4.22e2)
IGP-M	240.1403 (71.2655)	317.8180 (91.9919)	345.8193 (19.1527)	32.3917 (38.3086)	75.9227 (8.2931)	34.8366 (35.4442)	82.7605 (8.9731)
IGP-C	232.1166 (73.6961)	289.5270 (88.1854)	354.9536 (29.9707)	29.2240 (21.7221)	84.9971 (9.0191)	38.9569 (47.4119)	92.3965 (10.0547)

module is also the key component that enables IGP to deal with the *online* GP with non-fixed K. In our future work, we intend to further reduce the runtime of IGP by replacing the downstream clustering module with a generic E2E module that can directly derive the partitioning result with respect to the specific (non-fixed) K via an output layer.

- In most cases, IGP-M and IGP-C have similar quality metrics. On datasets in which the clustering structure is more difficult to be identified (e.g., $L(f, 0.6)$ and $L(n, 0.6)$), IGP-M has better partitioning quality than that of IGP-C. It implies that the modularity matrix $\mathbf{Q}_t$ is more informative than the normalized adjacency matrix $\mathbf{M}_t$ to be the neighbor similarity feature $\mathbf{X}_t$ in IGP. Moreover, the neighbor-induced feature $\mathbf{X}_t$ is also the key component to ensure the strong robustness of IGP, especially when the clustering structure of a given graph is difficult to be identified.

C.5.2 Trade-Off Analysis

In this study, we focus on the trade-off between quality and efficiency of the *online* GP on newly generated graphs from a system. To the best of our knowledge, to evaluate the trade-off between two aspects (that conflict with each other) using a comprehensive metric remains an open issue. We propose a *trade-off score* (TOS) to quantitatively evaluate the trade-off between quality and efficiency of a given GP method. Concretely, the calculation of TOS is based on the (normalized) area covered by an induced rectangle of a data point in the quality-efficiency visualization result (like Fig. 3). Fig. 4 gives two running examples regarding how to calculate TOS for a given GP method. Our goal is to let larger TOS indicate better a trade-off between quality and efficiency.

Table 10: Quantitative evaluation results in terms of NCut (↓) on datasets without ground-truth.

	Taxi	*Reddit*	*AS*	*Enron*
SNMF	0.7851 (0.1299)	6.7052 (8.4353)	10.0464 (1.6011)	23.0720 (1.34e2)
DCSBM	0.7908 (0.1338)	7.22e3 (3.89e3)	4.76e2 (3.74e2)	3.05e3 (2.10e3)
SC	0.7677 (0.1294)	2.8457 (1.5721)	2.2442 (0.2929)	0.7977 (0.5563)
DeepWalk	0.7999 (0.1198)	56.7501 (3.51e2)	6.9109 (1.1831)	54.3684 (2.48e2)
node2vec	0.8069 (0.1201)	4.4548 (4.0488)	6.1055 (1.0540)	14.9095 (87.0920)
LINE	0.7685 (0.1260)	9.31e2 (1.45e3)	2.98e2 (5.46e2)	1.04e2 (4.30e2)
VERSE	0.7983 (0.1247)	1.26e3 (1.53e4)	11.8885 (9.2661)	1.48e2 (6.50e2)
MILE	0.8378 (0.1261)	9.98e2 (1.56e3)	8.6199 (2.3743)	1.37e2 (6.10e2)
PhUSION	0.8541 (0.1372)	40.3041 (69.9684)	6.2456 (1.0237)	91.8462 (1.35e2)
NRP	51.7825 (57.8389)	6.84e2 (7.13e2)	1.72e3 (1.15e3)	4.66e2 (5.48e2)
RandNE	0.8050 (0.1239)	5.33e3 (3.50e3)	6.62e3 (3.08e3)	3.26e3 (2.35e3)
ProNE	0.7824 (0.1251)	52.6408 (4.52e2)	12.8754 (4.0178)	35.1600 (1.09e2)
Metis	1.0695 (0.1264)	2.94e2 (4.45e2)	15.3312 (3.7055)	2.95e2 (4.39e2)
hMetis	0.8305 (0.1157)	6.76e2 (7.88e2)	39.3087 (1.59e2)	4.26e2 (6.40e2)
GraClus	0.8493 (0.1195)	2.20e3 (1.75e3)	17.8574 (6.7540)	3.08e2 (7.82e2)
GraSAGE	12.4076 (2.1775)	6.59e3 (3.97e3)	1.05e4 (3.85e3)	4.59e3 (1.68e3)
GAT	1.20e3 (4.26e2)	5.21e3 (2.86e3)	5.27e3 (1.84e3)	2.17e3 (9.53e2)
GIN	25.31 (4.63)	6.16e3 (3.74e3)	1.18e4 (3.01e3)	6.34e3 (2.60e3)
GAP	0.9371 (0.1529)	4.1535 (4.4771)	6.1307 (1.2355)	1.3524 (0.4140)
ClusNet	6.9216 (42.5113)	28.2975 (53.4356)	31.4675 (49.0738)	23.6067 (99.4427)
IGP-M	0.7711 (0.1241)	4.7408 (2.1934)	7.2568 (1.2706)	4.5338 (16.9436)
IGP-C	0.7736 (0.1253)	4.7084 (2.1537)	9.6586 (3.4559)	4.3581 (16.5047)

Table 11: Quantitative evaluation results in terms of runtime (sec) (↓) on datasets w/ ground-truth.

	GN-0.5	*GN*-0.4	*GN*-0.3	*L*(*f*,0.3)	*L*(*f*,0.6)	*L*(*n*,0.3)	*L*(*n*,0.6)
SNMF	216.2452 (63.4919)	232.3165 (51.1009)	326.8980 (63.6736)	129.1531 (32.0298)	136.2061 (0.6368)	155.8362 (43.8659)	194.1315 (54.3392)
DCSBM	143.9118 (14.9216)	143.4681 (16.8809)	155.9920 (19.9667)	123.0517 (7.9505)	127.2608 (11.3991)	161.9426 (24.3826)	165.5730 (28.2101)
SC	49.3753 (0.6018)	46.7263 (0.6936)	62.1403 (1.0794)	21.0425 (1.2806)	42.5445 (2.2104)	26.9992 (3.8460)	53.2581 (6.9507)
DeepWalk	35.6981 (0.5680)	36.6256 (0.6653)	42.9227 (0.9463)	14.9011 (0.3784)	18.0380 (0.5342)	16.8692 (1.0360)	19.4530 (1.1855)
node2vec	54.3600 (0.5797)	54.6299 (0.7114)	59.7543 (1.0103)	30.9706 (0.6243)	33.5472 (0.6145)	34.8124 (2.0267)	35.8593 (2.0024)
LINE	34.2949 (1.7164)	29.3787 (1.9901)	24.1571 (1.7396)	8.1533 (0.7065)	8.5101 (0.6198)	10.8420 (1.8681)	13.2813 (1.5247)
VERSE	185.2756 (0.9992)	185.9548 (1.0422)	184.8650 (1.1709)	112.8455 (0.9151)	114.2033 (0.9718)	123.7247 (6.5278)	125.1043 (6.8006)
MILE	48.9463 (3.7566)	50.7684 (3.7649)	56.9629 (3.6718)	16.3114 (11.6159)	21.3616 (3.4539)	17.1403 (3.5605)	23.3779 (3.6040)
PhUSION	112.5309 (0.8327)	113.6472 (0.5111)	115.9772 (0.5784)	85.6189 (0.4948)	86.5876 (0.4538)	105.6061 (11.1439)	105.6130 (10.9538)
NRP	13.1071 (0.2830)	14.0291 (0.5710)	16.5823 (1.3357)	2.4441 (0.2301)	2.8182 (0.2380)	5.2406 (0.4901)	5.0703 (0.5747)
RandNE	10.3444 (0.0889)	12.4838 (0.1362)	14.7331 (0.2235)	2.1161 (0.1768)	2.4892 (0.2227)	2.5443 (0.2966)	2.9066 (0.3274)
ProNE	17.8259 (0.0987)	18.8484 (0.1309)	21.3583 (0.3047)	6.6678 (0.2314)	7.4628 (0.3266)	7.8484 (0.6965)	8.8083 (0.7115)
Metis	2.2119 (0.0556)	2.5956 (0.0446)	3.2307 (0.0802)	1.0741 (0.0737)	1.4749 (0.0508)	1.2073 (0.0989)	1.6377 (0.0935)
hMetis	12.6836 (0.4348)	13.5908 (0.4779)	15.5286 (0.5767)	5.5074 (0.2108)	7.5798 (0.2550)	6.2858 (0.5032)	8.6158 (0.6654)
GraClus	1.5905 (0.0837)	1.7698 (0.0736)	1.8823 (0.1018)	0.8802 (0.0821)	0.9393 (0.0356)	0.9528 (0.0441)	1.0287 (0.0489)
GraSAGE	8.1151 (0.2669)	8.2410 (0.3071)	8.1811 (0.2961)	2.5760 (0.1711)	2.4972 (0.1612)	3.1226 (0.3318)	3.3761 (0.3593)
GAT	5.6995 (0.1184)	5.5587 (0.1507)	5.3745 (0.1169)	2.0286 (0.1557)	2.3695 (0.1866)	2.7747 (0.3072)	2.8086 (0.2850)
GIN	4.4933 (0.0685)	4.5831 (0.1034)	4.5572 (0.0561)	1.8595 (0.1370)	1.9316 (0.1465)	1.8349 (0.2165)	1.8523 (0.2321)
GAP	19.6257 (0.6619)	19.5947 (0.5944)	19.5901 (0.6105)	66.4503 (1.9600)	66.9368 (1.7523)	76.7907 (6.7279)	77.6622 (6.5734)
ClusNet	38.6638 (0.2281)	39.2087 (0.2115)	40.1573 (0.2356)	73.2750 (4.9692)	76.9597 (4.2664)	88.5002 (7.6256)	90.0728 (8.1367)
IGP-M	9.9305 (0.2499)	11.1947 (0.3210)	13.6277 (0.5464)	2.4774 (0.1762)	3.5589 (0.2622)	2.9269 (0.3240)	4.1897 (0.4612)
Feat	0.4183 (0.0120)	0.4183 (0.0091)	0.4246 (0.0113)	0.1974 (0.0086)	0.1970 (0.0080)	0.2265 (0.0188)	0.2286 (0.0189)
Prop	0.0195 (0.0016)	0.0199 (0.0010)	0.0203 (0.0005)	0.0306 (0.0006)	0.0305 (0.0005)	0.0348 (0.0023)	0.0345 (0.0023)
Clus	9.4927 (0.2470)	10.7565 (0.3206)	13.1828 (0.5454)	2.2494 (0.1766)	3.3314 (0.2620)	2.6656 (0.3091)	3.9266 (0.4461)
IGP-C	9.9296 (0.2811)	11.4214 (0.2999)	14.1133 (0.5049)	2.8410 (0.1951)	3.5533 (0.2521)	3.3588 (0.3693)	4.2413 (0.4419)
Feat	0.4559 (0.0095)	0.4602 (0.0103)	0.47142 (0.0121)	0.2361 (0.0090)	0.2454 (0.0090)	0.2758 (0.0255)	0.2798 (0.0261)
Prop	0.0193 (0.0018)	0.0137 (0.0003)	0.0204 (0.0003)	0.0306 (0.0004)	0.0305 (0.0004)	0.0339 (0.0024)	0.0338 (0.0023)
Clus	9.4544 (0.2802)	10.9475 (0.2989)	13.62177 (0.5039)	2.5740 (0.1953)	3.2774 (0.2511)	3.0491 (0.3490)	3.9277 (0.4207)

In our evaluation, we use NMI, AC, modularity, and NCut as the quality metrics, while we evaluate the efficiency of a specific method via its total runtime. Let Q and E be the value of a quality metric and the runtime with respect to the method to be evaluated. Given the tuple (E, Q), each method used in our experiments can be visualized as a data point in a 2-D space as illustrated in Fig. 3. When we use NMI or AC as the quality metric Q, we further derive the normalized metric via

$$\widehat{Q} = Q/1, \tag{15}$$

since the value range of NMI or AC is $Q \in [0, 1]$ and large NMI or AC indicate better partitioning quality. Furthermore, when we use modularity as the quality metric, its value range is $Q \in [-1, 1]$, in which large modularity indicates better quality. Hence, we derive the normalized metric as

$$\widehat{Q} = (Q + 1)/2. \tag{16}$$

In contrast, smaller NCut metric implies better partitioning quality with the value range $Q \in [0, +\infty)$. Therefore, when using NCut as the quality metric Q, we normalize it via

$$\widehat{Q} = (Q_m - Q)/Q_m \tag{17}$$

with Q_m as the maximum NCut value through all the methods to be evaluated. For the efficiency metric E (i.e., runtime), smaller value of E indicates higher efficiency with the value range $E \in$

Table 12: Quantitative evaluation results in terms of runtime (sec) (↓) on datasets w/o ground-truth.

	Taxi	*Reddit*	*AS*	*Enron*
SNMF	2.1318 (1.4017)	217.1871 (372.5194)	112.3132 (83.5330)	1.6809 (1.2530)
DCSBM	6.1534 (0.1540)	9.3435 (12.6326)	197.1677 (135.8869)	3.1707 (1.5424)
SC	1.3371 (0.0840)	0.3480 (0.5234)	15.1301 (9.6621)	0.2766 (0.3300)
DeepWalk	8.9658 (0.0902)	9.9875 (6.0518)	34.3312 (14.7125)	5.6953 (1.7378)
node2vec	19.7044 (0.3069)	25.3689 (17.8262)	29.9006 (14.2057)	4.2710 (1.4861)
LINE	0.9986 (0.0764)	2.3979 (0.7268)	3.7648 (1.1209)	1.3753 (0.1353)
VERSE	20.1730 (0.1553)	25.3862 (11.4300)	98.6125 (40.1892)	20.1842 (0.1686)
MILE	11.3889 (5.4848)	6.5609 (2.7895)	8.4945 (2.3642)	2.5084 (0.6577)
PhUSION	6.4745 (0.0691)	4.9237 (5.5433)	77.7417 (49.3501)	2.5341 (1.2439)
NRP	0.7482 (0.0551)	0.3554 (0.2031)	1.5764 (0.6263)	0.1985 (0.1851)
RandNE	0.1820 (0.0159)	0.4783 (0.2712)	0.7422 (0.3132)	0.2618 (0.0728)
ProNE	0.9874 (0.0280)	0.7608 (0.5117)	4.8856 (2.6473)	0.4185 (0.1272)
Metis	0.2198 (0.0063)	0.1386 (0.0596)	0.3692 (0.1063)	0.0964 (0.0233)
hMetis	2.2233 (0.1212)	0.4217 (0.2423)	2.5954 (1.2534)	0.3171 (0.1250)
GraClus	1.0221 (0.0123)	0.1814 (0.0508)	0.4167 (0.1464)	0.1416 (0.0255)
GraSAGE	0.2077 (0.0156)	0.3549 (0.1880)	1.2886 (0.6155)	0.1738 (0.0411)
GAT	0.2118 (0.0232)	0.3282 (0.1626)	0.6855 (0.3030)	0.1364 (0.0288)
GIN	0.1598 (0.0130)	0.2409 (0.1024)	0.6653 (0.2863)	0.1332 (0.0279)
GAP	8.6571 (2.5928)	6.3181 (4.4778)	70.5331 (45.1668)	3.1985 (0.8925)
ClusNet	19.0952 (0.2832)	6.5504 (3.9219)	49.9295 (29.6607)	4.8045 (1.1453)
IGP-M	0.4323 (0.0163)	0.2933 (0.1374)	2.3692 (1.2549)	0.2094 (0.0599)
Feat	0.2399 (0.0036)	0.0066 (0.0077)	1.1176 (0.7288)	0.0259 (0.0148)
Prop	0.0027 (0.0001)	0.0126 (0.0085)	0.1277 (0.0524)	0.0026 (0.0011)
Clus	0.1897 (0.0154)	0.2741 (0.1242)	1.1239 (0.4813)	0.1809 (0.0460)
IGP-C	0.4268 (0.0156)	0.2971 (0.1395)	2.3835 (1.2837)	0.2222 (0.0620)
Feat	0.2353 (0.0049)	0.0082 (0.0093)	1.1558 (0.7496)	0.0264 (0.0146)
Prop	0.0027 (0.0001)	0.0137 (0.0087)	0.1273 (0.0519)	0.0026 (0.0011)
Clus	0.1888 (0.0145)	0.2752 (0.1243)	1.1004 (0.4865)	0.1932 (0.0489)

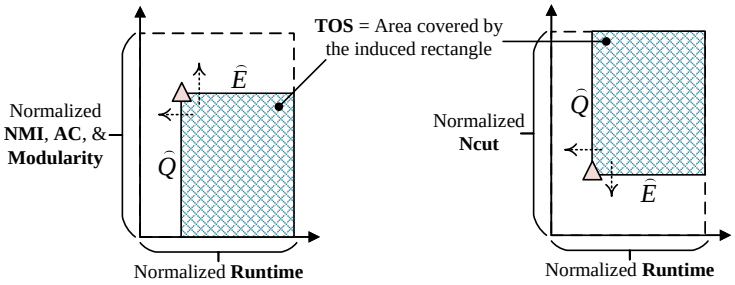


Figure 4: Running examples of the calculation of *trade-off score* (TOS).

$[0, +\infty)$, so we further normalize E via

$$\widehat{E} = (E_m - E)/E_m, \tag{18}$$

where E_m denotes the maximum runtime through all the methods to be evaluated. Finally, we define TOS with respect to the given quality and efficiency metrics (Q, E) as the (normalized) area covered by the induced rectangle illustrated in Fig. 4. Namely, we have

$$\mathrm{TOS}(Q, E) = \widehat{Q} \times \widehat{E}, \tag{19}$$

in which larger TOS implies a better trade-off between quality and efficiency.

The TOS values with respect to the quality metrics of NMI, AC, modularity, and NCut are depicted in Table 13, Table 14, Table 15, and Table 16, respectively. In our experiments, we saved the model parameters of IGP with best metrics in terms of NMI and modularity on the validation sets of datasets with and without ground-truth, respectively. Namely, we use NMI and modularity as the validation metrics for datasets with and without ground-truth. For the TOS metrics with respect to NMI and modularity (see Table 13 and Table 15), both IGP-M and IGP-C have the best and second-best TOS values in most cases, indicating that IGP can achieve the best trade-off when using NMI and modularity as the quality metrics. Furthermore, IGP-M and IGP-C can achieve the top-5 TOS metrics with respect to AC and NCut on all the datasets. In summary, IGP can achieve a better trade-off between quality and efficiency of the *online* GP over various baseline methods.

Table 13: Trade-off scores (↑) between NMI and runtime on datasets with ground-truth.

	GN-0.3	*GN*-0.4	*GN*-0.5	*L*(*f*,0.3)	*L*(*f*,0.6)	*L*(*n*,0.3)	*L*(*n*,0.6)
SNMF	0	0	0	0	0	0.031519	0
DCSBM	0.260151	0.220595	0.227190	0.021561	0.017910	0	0.040190
SC	0.763420	0.788722	0.769741	0.821922	0.246040	0.818364	0.178948
DeepWalk	0.719368	0.822130	0.829909	0.825089	0.483322	0.836349	0.507484
node2vec	0.721268	0.756357	0.744426	0.723333	0.445514	0.747508	0.488192
LINE	0.493149	0.600122	0.788651	0.756242	0.400227	0.745414	0.361269
VERSE	0.265689	0.169129	0.131887	0.113387	0.072645	0.212255	0.163100
MILE	0.771165	0.774983	0.772029	0.820147	0.171753	0.841492	0.181809
PhUSION	0.605796	0.504730	0.477024	0.319715	0.197372	0.330138	0.253156
NRP	0.739484	0.852510	0.892231	0.547244	0.260496	0.502205	0.216786
RandNE	0.756114	0.890245	0.936262	0.566858	0.182306	0.571675	0.185875
ProNE	0.903072	0.914641	0.915731	0.885116	0.528183	0.887878	0.549961
Metis	0.800609	0.936222	0.976706	0.842336	0.440379	0.847931	0.451459
hMetis	0.781810	0.883314	0.906516	0.850612	0.393322	0.856704	0.407476
GraClus	0.901976	0.973527	0.988079	0.899627	0.526941	0.900669	0.536144
GraSAGE	0.421286	0.443489	0.440428	0.220708	0.155005	0.241943	0.163703
GAT	0.438176	0.439526	0.448752	0.226289	0.152205	0.254661	0.156010
GIN	0.428837	0.424066	0.423905	0.182238	0.131311	0.174105	0.133019
GAP	0.527663	0.589407	0.660929	0.455052	0.240550	0.492373	0.288036
ClusNet	0.114907	0.146712	0.215977	0.388693	0.241368	0.408611	0.300387
IGP-M	0.922471	0.945911	0.952646	0.953453	0.546050	0.955414	0.552611
Ranking	1	2	4	1	1	1	1
IGP-C	0.913769	0.945702	0.952651	0.943186	0.507506	0.945573	0.515584
Ranking	2	3	3	2	4	2	4

Table 14: Trade-off scores (↑) between AC and runtime on datasets with ground-truth.

	GN-0.3	*GN*-0.4	*GN*-0.5	*L*(*f*,0.3)	*L*(*f*,0.6)	*L*(*n*,0.3)	*L*(*n*,0.6)
SNMF	0	0	0	0	0	0.027511	0
DCSBM	0.127377	0.098786	0.127377	0.010558	0.008965	0	0.019183
SC	0.761561	0.772266	0.761561	0.795973	0.178857	0.810531	0.088966
DeepWalk	0.804861	0.784645	0.804861	0.777673	0.476902	0.785197	0.495967
node2vec	0.722267	0.730658	0.722267	0.684258	0.440162	0.706215	0.479305
LINE	0.725041	0.409690	0.725041	0.675859	0.358320	0.658267	0.313292
VERSE	0.115661	0.139734	0.115661	0.105924	0.065020	0.197199	0.145890
MILE	0.763364	0.757321	0.763364	0.809051	0.085919	0.827722	0.087430
PhUSION	0.462780	0.484042	0.462780	0.300468	0.188811	0.308917	0.240525
NRP	0.766634	0.684977	0.766634	0.414701	0.199975	0.374960	0.160009
RandNE	0.905222	0.819086	0.905222	0.400036	0.126741	0.400015	0.126084
ProNE	0.905546	0.897366	0.905546	0.782977	0.491887	0.782448	0.510821
Metis	0.968986	0.902305	0.968986	0.582118	0.347496	0.581433	0.350716
hMetis	0.828479	0.795002	0.828479	0.637792	0.309747	0.634286	0.317839
GraClus	0.968226	0.943854	0.968226	0.715093	0.458715	0.713875	0.462536
GraSAGE	0.075362	0.073883	0.075362	0.073994	0.056838	0.074338	0.055616
GAT	0.089575	0.084528	0.089575	0.083862	0.061118	0.086296	0.056077
GIN	0.074519	0.073618	0.074519	0.068696	0.057473	0.060111	0.053188
GAP	0.353696	0.272041	0.353696	0.406017	0.203730	0.436742	0.240460
ClusNet	0.037036	0.024438	0.037036	0.374026	0.244457	0.393963	0.301512
IGP-M	0.944155	0.926685	0.944155	0.924813	0.521703	0.925367	0.522671
Ranking	3	3	3	1	1	1	1
IGP-C	0.943873	0.926876	0.943873	0.905435	0.461440	0.905423	0.464329
Ranking	4	2	4	2	4	2	5

C.5.3 CONVERGENCE ANALYSIS

We also analyzed the convergence of the *offline* training procedure of IGP (i.e., Algorithm 3). In each epoch of the *offline* training on a dataset, we recorded the average partitioning quality (in terms of NMI, AC, modularity, and NCut) with respect to the learned embedding $\{\mathbf{U}_t\}$ (i) on the training set Γ_{T} and validation set Γ_{V} (i.e., historical system snapshots). Moreover, we also evaluated the average partitioning quality of the generalized embedding $\{\mathbf{U}_t\}$ on the test set (i.e., new snapshots) in each epoch. To further validate the effectiveness of the auxiliary *label-induced embedding* $\{\mathbf{U}_t^{(g)}\}$, the average partitioning quality of $\{\mathbf{U}_t^{(g)}\}$ on the training set was also recorded in each epoch during the *offline* training. Note that we don't need to derive $\mathbf{U}_t^{(g)}$ for the validation set Γ_{V} and test set Γ'.

We use the evaluation records on the challenging *L*(*n*, 0.6) dataset, whose clustering structures are difficult to be identified, as an example. The convergence curves (within the first 50 epochs) of the average NMI, AC, modularity, and NCut with respect to (i) $\{\mathbf{U}_t^{(g)}\}$ on training set Γ_{T}, (ii) $\{\mathbf{U}_t\}$ on training set Γ_{T}, (iii) $\{\mathbf{U}_t\}$ on validaiton set Γ_{V}, and (iv) $\{\mathbf{U}_t\}$ on test set Γ' for IGP-M and IGP-C are illustrated in Fig. 5 and Fig. 6, respectively.

One can have similar observations in Fig. 5 and Fig. 6. Concretely, the partitioning quality (in terms of NMI, AC, modularity, and NCut) of $\{\mathbf{U}_t\}$ continuously improves in the first several epochs on the training, validation, and test set. More importantly, the average NMI and AC metrics of $\{\mathbf{U}_t^{(g)}\}$

Table 15: Trade-off scores (↑) between modularity and runtime on all the datasets.

	GN-0.3	*GN*-0.4	*GN*-0.5	*L*(*f*,0.3)	*L*(*f*,0.6)	*L*(*n*,0.3)	*L*(*n*,0.6)	*Taxi*	*Reddit*	*AS*	*Enron*
SNMF	0	0	0	0	0	0.029417	0	0.776810	0	0.329919	0.796998
DCSBM	0.196199	0.209064	0.196199	0.026306	0.034404	0	0.077019	0.603615	0.491935	0	0.439831
SC	0.571537	0.550460	0.571537	0.700295	0.435418	0.697871	0.448530	0.811401	0.804659	0.660779	0.845404
DeepWalk	0.617339	0.578144	0.617339	0.717917	0.580924	0.727192	0.603132	0.482555	0.790210	0.646167	0.627100
node2vec	0.553566	0.527400	0.553566	0.620439	0.506752	0.640979	0.548849	0.020167	0.747667	0.671002	0.697181
LINE	0.599377	0.506479	0.599377	0.730197	0.578169	0.724094	0.562492	0.826696	0.777569	0.711255	0.796463
VERSE	0.101203	0.128339	0.101203	0.101328	0.104468	0.189411	0.230053	0	0.669842	0.380514	0
MILE	0.573316	0.539526	0.573316	0.726791	0.511507	0.744610	0.534739	0.376458	0.771421	0.750367	0.765296
PhUSION	0.354651	0.351922	0.354651	0.275137	0.243710	0.283974	0.305638	0.587142	0.819442	0.479054	0.733840
NRP	0.682043	0.630997	0.682043	0.609837	0.524273	0.604774	0.514648	0.510680	0.519199	0.518868	0.496271
RandNE	0.701221	0.641614	0.701221	0.582497	0.506864	0.583093	0.509161	0.857791	0.526438	0.524518	0.599768
ProNE	0.679917	0.634937	0.679917	0.763251	0.626863	0.765891	0.635973	0.825562	0.811547	0.727027	0.797122
Metis	0.728917	0.668645	0.728917	0.755564	0.645484	0.756369	0.647392	0.843112	0.761214	0.762320	0.821259
hMetis	0.681441	0.633723	0.681441	0.753967	0.622327	0.756645	0.630661	0.771624	0.769852	0.769881	0.824539
GraClus	0.733713	0.681469	0.733713	0.762468	0.653512	0.762636	0.655060	0.821411	0.725743	0.732499	0.804417
GraSAGE	0.481621	0.482456	0.481621	0.491399	0.491717	0.493105	0.492140	0.602977	0.533826	0.524996	0.542340
GAT	0.489158	0.489354	0.489158	0.497019	0.491695	0.499837	0.492865	0.500242	0.542629	0.521182	0.517479
GIN	0.489855	0.490283	0.489855	0.495265	0.493205	0.517717	0.515979	0.536169	0.523519	0.526318	0.550443
GAP	0.542591	0.503381	0.542591	0.404051	0.338092	0.438136	0.398847	0.490338	0.801389	0.508388	0.738320
ClusNet	0.410725	0.415697	0.410725	0.358061	0.293826	0.375891	0.362485	0.046092	0.753129	0.581992	0.657425
IGP-M	0.707210	0.657560	0.707210	0.815354	0.653175	0.817208	0.656616	0.851405	0.845756	0.774678	0.862607
Ranking	4	3	4	1	2	1	1	3	1	1	1
IGP-C	0.707213	0.656981	0.707213	0.810373	0.641857	0.811953	0.645336	0.851593	0.845692	0.767607	0.861164
Ranking	3	4	3	2	4	2	4	2	2	3	2

Table 16: Trade-off scores (↑) between NCut and runtime on all the datasets.

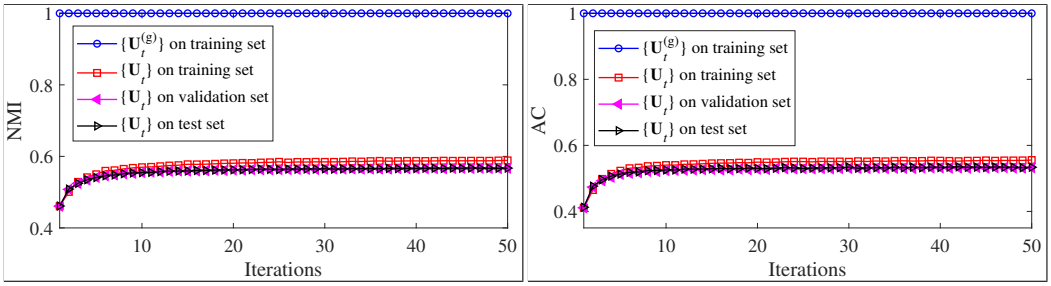
	GN-0.3	*GN*-0.4	*GN*-0.5	*L*(*f*,0.3)	*L*(*f*,0.6)	*L*(*n*,0.3)	*L*(*n*,0.6)	*Taxi*	*Reddit*	*AS*	*Enron*
SNMF	0	0	0	0	0	0.037673	0	0.893742	0	0.430002	0.913388
DCSBM	0.463741	0.368225	0.327831	0.042735	0.061690	0	0.140382	0.694513	0	0	0.437516
SC	0.808108	0.797757	0.770458	0.836208	0.685506	0.832580	0.724499	0.933124	0.998005	0.923088	0.986172
DeepWalk	0.862738	0.840345	0.833658	0.883687	0.865694	0.895040	0.898328	0.555186	0.946523	0.825396	0.711682
node2vec	0.813678	0.763560	0.747722	0.758979	0.752068	0.784204	0.813995	0.023214	0.882649	0.847912	0.786546
LINE	0.907770	0.868226	0.839728	0.866537	0.881759	0.877748	0.898793	0.949892	0.861512	0.956148	0.916469
VERSE	0.431836	0.199113	0.142938	0.126107	0.161129	0.235760	0.354895	0	0.728985	0.499353	0
MILE	0.823834	0.780472	0.772657	0.872764	0.841790	0.893535	0.878524	0.435136	0.835769	0.956221	0.856717
PhUSION	0.643706	0.510255	0.479205	0.336830	0.363561	0.347682	0.455305	0.678570	0.971880	0.605388	0.861791
NRP	0.913374	0.929636	0.935351	0.926295	0.781900	0.889897	0.803601	0.921556	0.903878	0.847285	0.917284
RandNE	0.923444	0.939974	0.949806	0.918973	0.634815	0.925991	0.731656	0.990316	0.261044	0.438627	0.479630
ProNE	0.930435	0.917548	0.916470	0.947070	0.943203	0.950357	0.953153	0.950436	0.989239	0.974161	0.973838
Metis	0.986884	0.987660	0.989027	0.990565	0.986847	0.991656	0.989854	0.988227	0.958648	0.996836	0.948820
hMetis	0.949211	0.940104	0.940398	0.955406	0.942396	0.959568	0.954172	0.889175	0.904619	0.983561	0.918185
GraClus	0.991680	0.991273	0.991850	0.992037	0.990974	0.993202	0.993134	0.948665	0.693918	0.996382	0.944724
GraSAGE	0.074026	0	0	0	0	0	0	0.979519	0.086931	0.110296	0.273798
GAT	0.108244	0.525741	0.574205	0.461489	0.142573	0.312092	0.375534	0	0.277902	0.553032	0.652783
GIN	0	0.287524	0.267366	0.288281	0.249078	0.069530	0.262080	0.971253	0.146522	0	0
GAP	0.861703	0.880083	0.887997	0.485246	0.507609	0.525614	0.599120	0.570413	0.970351	0.641936	0.841355
ClusNet	0.711201	0.717848	0.576839	0.428888	0.421953	0.450751	0.524419	0.053121	0.966043	0.744782	0.759132
IGP-M	0.956159	0.950342	0.952857	0.979890	0.971879	0.981212	0.976948	0.977945	0.997994	0.987379	0.988918
Ranking	3	3	4	3	3	3	3	5	2	3	1
IGP-C	0.954620	0.949499	0.952902	0.977168	0.971681	0.978463	0.976511	0.978215	0.997981	0.987106	0.988312
Ranking	4	4	3	4	4	4	4	4	3	4	2

on training set Γ_{T} are always 1 in all the epochs for both IGP-M and IGP-C, which indicates that the partitioning results derived from the auxiliary *label-induced* embedding $\{\mathbf{U}_t^{(g)}\}$ have a perfect mapping to the ground-truth. For both IGP-M and IGP-C, the average modularity and NCut metrics of $\{\mathbf{U}_t^{(g)}\}$ also keep at the values that are much better than the metrics of $\{\mathbf{U}_t\}$ on training, validation, and test set. The convergence curves with respect to $\{\mathbf{U}_t^{(g)}\}$ further verify that the introduced *label-induced* embedding $\{\mathbf{U}_t^{(g)}\}$ can capture much more information regarding the partitioning results (e.g., ground-truth) of training set (i.e., historical system snapshots) than the original learned embedding $\{\mathbf{U}_t\}$, which enables IGP to capture the permutation invariant partitioning labels during the *offline* training. It ensures the strong robustness of IGP beyond existing baselines.

In summary, the convergence curves in Fig. 5 and Fig. 6 validate the effectiveness of the *offline* training procedure and the proposed dual GNN structure of IGP.

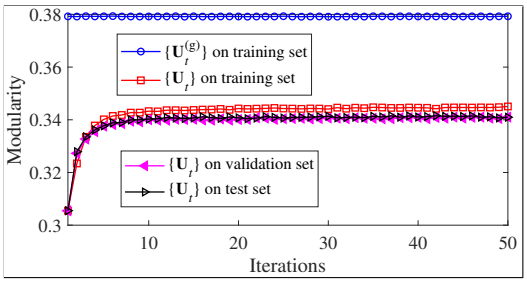
C.5.4 Ablation Study

In addition to the ablation study results shown in Table 2, we conducted additional ablation studies on *GN*-0.3, *L*(*f*,0.6), and *L*(*n*,0.6), where the clustering structures are more difficult to be identified compared with other datasets. In each case of the ablation study, the mean value μ and standard deviation σ of all the quality metrics (on the test set Γ') were recorded. The detailed ablation study

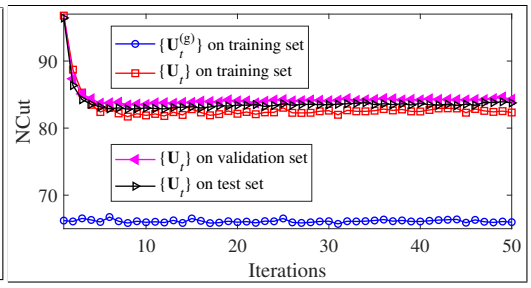


(a) Convergence curves of average NMI

(b) Convergence curves of average AC

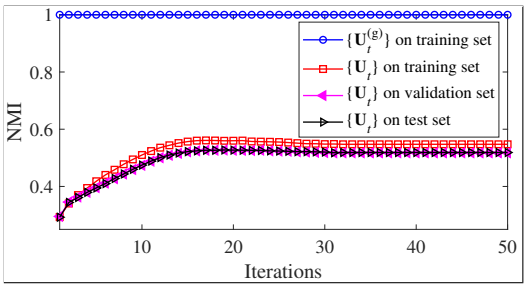


(c) Convergence curves of average modularity

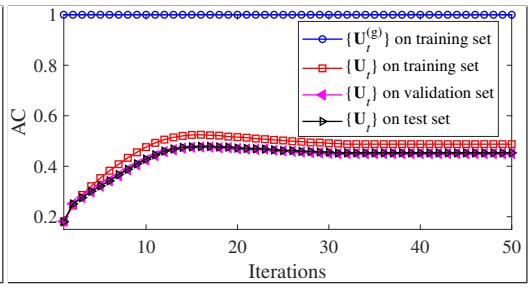


(d) Convergence curves of average NCut

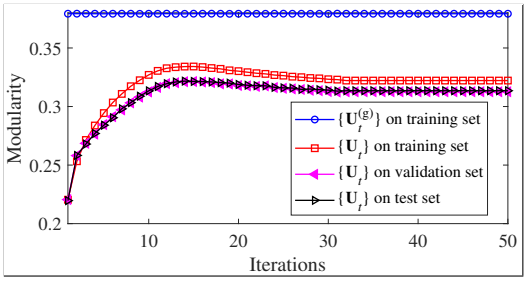
Figure 5: Convergence curves of IGP-M on $L(n,.6)$ in terms of NMI, AC, modularity, and NCut.



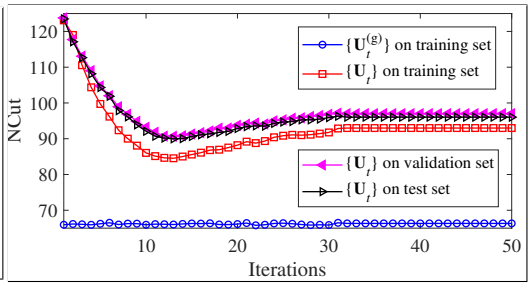
(a) Convergence curves of average NMI



(b) Convergence curves of average AC



(c) Convergence curves of average modularity



(d) Convergence curves of average NCut

Figure 6: Convergence curves of IGP-C on $L(n,.6)$ in terms of NMI, AC, modularity, and NCut.

results on $L(f,0.6)$, $L(n,0.6)$, and GN-0.3 (in the format of $\mu(\sigma)$) are illustrated in Table 17, Table 18, and Table 19, respectively.

Note that Table 19, Table 17, and Table 18 have consistent results. The FR loss, feature input $\mathbf{Z}_t$, and GNN are key components to ensure the high quality of IGP, because there are significant quality declines in cases without the three components. Moreover, the AL and CR loss are the components to further enhance the partitioning quality, as they can incorporating the permutation invariant label information (e.g., the partitioning ground-truth) of historical snapshots to the *offline* training of IGP.

Table 17: Detailed ablation study on $L(f,.6)$.

	NMI (%) (↑)		**AC** (%) (↑)		**Modularity** (%) (↑)		**Ncut** (↓)	
	IGP-M	**IGP-C**	**IGP-M**	**IGP-C**	**IGP-M**	**IGP-C**	**IGP-M**	**IGP-C**
IGP	**56.07** (2.33)	**52.11** (2.79)	**53.57** (2.21)	**47.38** (2.68)	**34.14** (0.53)	**31.81** (0.65)	**75.9227** (8.2931)	**84.9971** (9.0191)
w/o AL	51.73 (2.86)	34.83 (3.29)	49.45 (2.59)	26.64 (2.79)	33.27 (0.63)	26.88 (0.80)	77.0460 (7.3274)	101.7889 (8.6091)
w/o FR	11.89 (1.80)	15.60 (1.75)	6.57 (0.34)	8.25 (0.55)	9.14 (1.38)	5.73 (0.62)	6.90e2 (1.59e2)	1.02e3 (8.22e2)
w/o CR	49.50 (3.06)	34.84 (3.25)	46.83 (2.84)	26.57 (2.65)	32.61 (0.72)	26.90 (0.76)	78.9251 (7.1861)	102.3065 (8.7022)
w/o $\mathbf{Z}_t$	14.13 (1.65)	15.39 (1.81)	5.71 (0.13)	5.79 (0.14)	0.24 (0.15)	0.76 (0.19)	2.62e4 (4.49e3)	1.03e4 (4.42e3)
w/o GNN	1.03 (0.20)	8.75 (0.55)	4.22 (0.15)	5.17 (0.15)	0.01 (0.01)	0.86 (0.23)	1.70e3 (4.99e2)	4.01e4 (7.24e3)

Table 18: Detailed ablation study on $L(n,.6)$.

	NMI (%) (↑)		**AC** (%) (↑)		**Modularity** (%) (↑)		**NCut** (↓)	
	IGP-M	**IGP-C**	**IGP-M**	**IGP-C**	**IGP-M**	**IGP-C**	**IGP-M**	**IGP-C**
IGP	**56.48** (2.08)	**52.71** (2.52)	**53.42** (2.04)	**47.47** (2.59)	**34.22** (0.47)	**31.95** (0.62)	**82.7605** (8.9731)	**92.3965** (10.0547)
w/o AL	52.13 (2.52)	35.12 (3.01)	49.30 (2.41)	26.15 (2.61)	33.35 (0.58)	26.70 (0.74)	84.0540 (8.6339)	113.0982 (10.2328)
w/o FR	11.19 (2.04)	15.46 (1.68)	6.28 (0.45)	5.46 (0.18)	10.08 (1.77)	0.45 (0.11)	1.01e3 (237.6611)	2.08e4 (4.79e3)
w/o CR	49.97 (2.79)	35.16 (2.92)	46.70 (2.73)	26.22 (2.50)	32.70 (0.68)	26.78 (0.71)	86.0849 (8.7142)	111.6198 (10.3943)
w/o $\mathbf{Z}_t$	16.39 (1.59)	13.88 (1.39)	6.14 (0.26)	5.57 (0.18)	1.27 (0.47)	0.28 (0.15)	3.01e4 (4.76e3)	2.88e4 (5.25e3)
w/o GNN	1.14 (0.25)	8.45 (0.45)	3.92 (0.18)	4.78 (0.22)	0.01 (0.02)	0.32 (0.32)	1.08e4 (2.86e3)	5.12e4 (7.94e3)

D EXTENDED APPLICATIONS

D.1 MODEL SELECTION

For the *online* GP in which the number of clusters K isn't given, IGP also has the potential to estimate the proper K value for a given graph. As a demonstration, we applied XMeans, a model selection method, to the derived embedding $\{\mathbf{U}_t\}$ of both IGP-M and IGP-C to estimate K for each snapshot in the test set of $L(f,.3)$, $L(f,.6)$, $L(n,.3)$ and $L(n,.6)$. In the 4 datasets, ground-truth regarding the K value is given by the dataset and K is non-fixed for the multiple test graphs. With the increase of μ, the clustering structures of $L(f,\mu)$ and $L(n,\mu)$ are increasingly difficult to be identified. In particular, we use the same parameter settings and layer configurations with that in Table 4. The range of K of XMeans for each dataset was set according to the maximum and minimum number of clusters as illustarted in Table 3. The model selection results and ground-truth with respect to the test sets of $L(n,\mu)$ and $L(n,\mu)$ are depicted in Fig. 7 and Fig 8, respectively.

The model selection results in Fig. 7 and Fig 8 are similar. When $\mu = 0.3$ (with *distinct clustering structure*), the model selection results of both IGP-M and IGP-C fit well to the ground-truth on most test snapshots of $L(f,0.3)$ and $L(n,0.3)$. Note that the potential of model selection of IGP is still largely influenced by the intrinsic clustering structure of the dataset. The estimated K values are still far from the ground-truth when $\mu = 0.6$ with *indistinct clustering structure* on $L(f,0.6)$ and $L(n,0.6)$. In summary, IGP has the potential to estimate the proper number of clusters K of newly generated graph snapshots with distinct clustering structures.

D.2 TRANSFERRING TEST

To test the potential of IGP to transfer from one (training) scenario to other different (generalization) scenarios, we consider the challenging application in which we first conduct the *offline* training of IGP on completely synthetic graphs and then directly generalize the model to the real datasets for fast *online* GP (without additional optimization).

In particular, we expect the synthetic graphs used for *offline* training cover various structural properties of real-world network systems. We adopt *LFR-Net* (described in Appendix C.2) as the model to generate the synthetic training graphs. In contrast to the parameter settings of *LFR-Net* in our quantitative evaluation (see Appendix C.2), we randomly set $N \in [500, 6000]$, $k \in [10, 100]$, $k_{\max} \in [50, 200]$, $c_{\min} \in [5, 100]$, $c_{\max} \in [50, 400]$, and $\mu \in [0.1, 0.7]$ to independently generate $2,000$ synthetic snapshots, where we ensure that $k_{\max} > k$ and $c_{\max} > c_{\min}$ for each snapshot.

In our experiments, we conducted the *offline* training on the aforementioned synthetic snapshots and generalized the trained model to the test set of the four real datasets used in our quantitative evaluation (i.e., *Taxi*, *Reddit*, *AS*, and *Enron* as introduced in Appendix C.2). We used the same layer configurations as in Table 4 with respect to each real dataset and tried different parameter settings of IGP-M and IGP-C to report their best average quality. The recommended parameter settings of IGP-M and IGP-C are shown in Table 20. The evaluation results on the test set of the

Table 19: Detailed ablation study on *GN*-0.3.

	NMI (%) (↑)		**AC** (%) (↑)		**Modularity** (%) (↑)		**NCut** (↓)	
	IGP-M	**IGP-C**	**IGP-M**	**IGP-C**	**IGP-M**	**IGP-C**	**IGP-M**	**IGP-C**
IGP	**96.26** (0.32)	**95.50** (0.43)	**91.96** (0.73)	**90.47** (0.85)	**27.92** (0.22)	**27.68** (0.25)	**345.8193** (19.1527)	**354.9536** (29.9707)
w/o AL	93.31 (0.57)	95.00 (0.41)	87.30 (1.02)	89.75 (0.85)	26.98 (0.30)	27.51 (0.25)	427.4439 (97.9730)	363.4415 (42.5595)
w/o FR	47.29 (0.69)	72.47 (0.93)	16.87 (0.57)	56.06 (1.46)	7.27 (0.27)	15.06 (0.43)	1.95e4 (2.84e3)	1.34e3 (827.2683)
w/o CR	93.75 (0.45)	95.03 (0.41)	87.85 (0.83)	89.86 (0.77)	27.11 (0.26)	27.52 (0.24)	404.9032 (85.2360)	371.1806 (83.1775)
w/o $\mathbf{Z}_t$	44.20 (0.18)	44.91 (0.17)	7.34 (0.12)	7.73 (0.12)	0.03 (0.03)	0.03 (0.03)	1.68e5 (1.28e4)	1.65e5 (1.32e4)
w/o GNN	30.08 (0.69)	56.25 (1.28)	12.99 (0.50)	33.66 (1.20)	4.53 (0.19)	11.80 (0.32)	6.11e3 (646.8439)	996.6852 (439.4099)

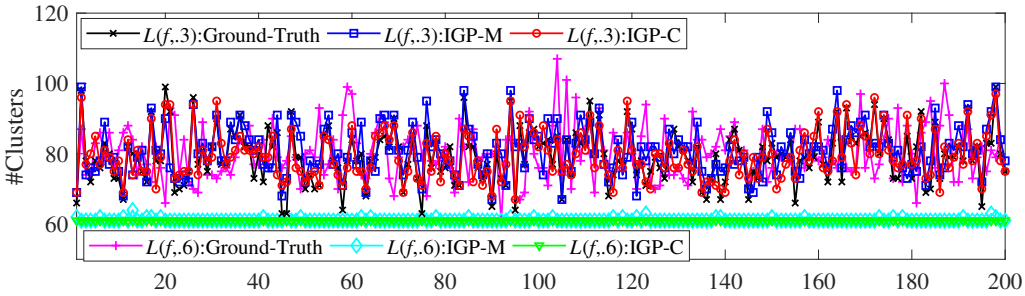


Figure 7: Model selection results of IGP-M and IGP-C on the test set of $L(f,0.3)$ and $L(f,0.6)$.

four real datasets (in terms of modularity and NCut) are depicted Table 21, where 'Real' and 'Syn.' represent the results with respect to (i) training on the training set of original real datasets (same as the results in Table 8 and Table 10) and (ii) training on the aforementioned synthetic snapshots, respectively. According to Table 21, we have the following key observations.

- In some cases (e.g., on *Taxi* and *Reddit*), the quality metrics of training on synthetic graphs are close to those of training on the original real datasets for both IGP-M and IGP-C, which are also competitive to the strong baselines as illustrated in Table 8 and Table 10. It preliminarily validates the potential of IGP to be transferred from completely synthetic graphs to real datasets.
- However, in other cases, there're still significant quality declines for both the variants of IGP, which indicate that the transferring capability of IGP is largely affected by the domain similarity between the synthetic training set and real test set. The negative transferring can be alleviated by carefully generating or selecting the synthetic snapshots with the assistant of the validation information from target real datasets. We leave the study of this application in our future work.
- Compared with IGP-C, IGP-M has better transferring quality metrics in most cases. The quality declines of IGP-C are also more significant than those of IGP-M. The results further validate that the modularity matrix $\mathbf{Q}_t$ used in IGP-M is a more informative neighbor-induced feature than the normalized adjacency matrix $\mathbf{M}_t$ used in IGP-C.

Note that IGP has similar motivations with SOTA pre-training methods for graphs, e.g., Pretrain-GNN (Hu et al., 2019) and GCC (Qiu et al., 2020), in which we pre-train a GNN-based model on historical graphs (in an *offline* way) and generalize it to new snapshots. The major difference between our IGP framework and SOTA pre-training methods is that a *fine-tuning* procedure with another optimization algorithm based on a certain fine-tuning loss on new test graphs is required for most pre-training methods after the *offline* (pre-)training. In particular, the *fine-tuning* procedure may still be time-consuming for new test graphs, where several iterative optimization algorithms (e.g., gradient descent) are used.

In this study, we focus on the NP-hard challenge of GP and try to achieve a better trade-off between quality and efficiency. To some extent, the proposed IGP framework sacrifices the opportunities to fine-tune the (pre-)trained model, which ensures the low runtime on new test graphs. In our future research, we intend to consider the promising application, in which we pre-train a high-quality GP model on completely synthetic graphs and fast fine-tune the (pre-)trained model on new unseen

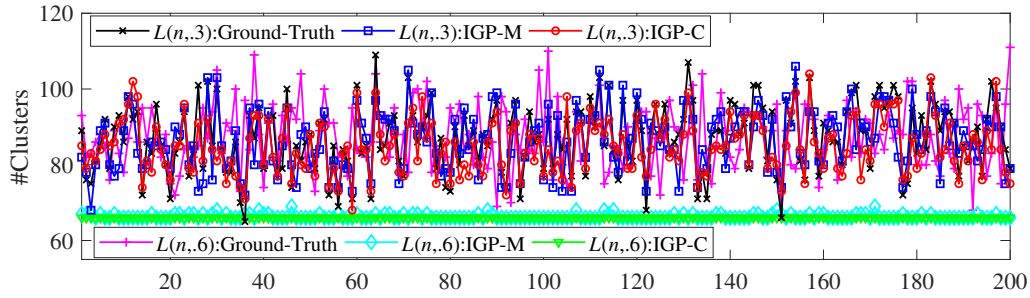


Figure 8: Model selection results of IGP-M and IGP-C on the test set of $L(n,0.3)$ and $L(n,0.6)$.

Table 20: Parameter setting of IGP in the transferring test.

	IGP-M		IGP-C	
	(α, β, m, p)	(η_G, η_D)	(α, β, m, p)	(η_G, η_D)
Taxi	(1, 1, 1, 1000)	(1e-4, 1e-4)	(1, 0.1, 1, 1000)	(1e-4, 1e-4)
Reddit	(1, 0.1, 1000)	(1e-4, 1e-4)	(1, 10, 1, 1000)	(5e-5, 5e-5)
AS	(1, 0.1, 1, 1000)	(1e-4, 1e-4)	(1, 5, 1, 1000)	(1e-4, 1e-4)
Enron	(1, 1, 1, 1000)	(1e-4, 1e-4)	(1, 0.1, 1, 1000)	(5e-5, 5e-5)

graphs (via several efficient fine-tuning strategies) with the guarantee of high partitioning quality and low runtime.

E Possible Future Work

We summarize several promising future research directions as follow.

- According to the results in Table 11 and Table 12, the downstream clustering module (i.e., KMeans with 10 independent runs) consumes most of the runtime of IGP. Although replacing the clustering module with an FC output layer (trained in an E2E way) to directly output the partitioning result can further reduce the total runtime, the downstream clustering module is the key component that enables IGP to handle the *online* GP with non-fixed K. In contrast, existing E2E GP methods still need to be trained from scratch for new graphs with non-fixed K. In our future research, we intend to further reduce the runtime of IGP by exploring a *generic E2E scheme* that can directly derive the GP result (e.g., via an output layer) and handle the *online* GP with non-fixed K. One possible solution is to simultaneously train multiple output layers (with different output dimensions) to derive multiple partitioning results with respect to all the possible values of K. Another possible strategy is to output the partitioning results in a hierarchical structure (that covers the results with respect to all the possible values of K) based on the idea of hierarchical clustering.
- As discussed in Appendix D.2, our IGP framework has similar motivation with SOTA pre-training methods for graphs (Hu et al., 2019; Qiu et al., 2020), i.e., conduct the *offline* (pre-)training of *inductive* GNNs (regardless of the time cost), but we sacrifice the opportunities to fine-tune IGP on new test graphs for a better trade-off between quality and efficiency of the *online* GP. The fine-tuning on new graphs (from other scenarios) can avoid the negative transferring by utilizing the validation information from the new targeted graphs. Moreover, to select the proper set of pre-training datasets is still an open issue in recent researches regarding the pre-training and fine-tuning scheme. In our future work, we intend to conduct the *offline* (pre-)training of IGP on completely synthetic graphs (that covers the properties of various real network systems) and explore an efficient strategy to fine-tune the model on real new system snapshots (of other scenarios) with the guarantee of high partitioning quality and low runtime.
- Scaling GNNs up to graphs with large number of nodes is a significant direction in recent researches, where several sampling strategies are used to split a large graph into multiple subgraphs (Chen et al., 2018; Huang et al., 2018; Zeng et al., 2019), i.e., mini-batches with small number of nodes. Note that IGP is an *inductive* framework across graphs, which can

Table 21: Transferring test results on real datasets.

		Modularity (%) (↑)				**NCut** (↓)			
		Taxi	*Reddit*	*AS*	*Enron*	*Taxi*	*Reddit*	*AS*	*Enron*
IGP-M	Real	74.01 (0.51)	69.38 (5.49)	56.82 (2.71)	74.33 (13.51)	0.7711 (0.1241)	4.7408 (2.1934)	7.2568 (1.2706)	4.5338 (16.9436)
	Syn.	74.04 (0.46)	69.31 (6.04)	54.71 (2.28)	73.96 (17.58)	0.7714 (0.1209)	4.1351 (1.8216)	7.2359 (1.2090)	18.4665 (98.8396)
IGP-C	Real.	74.00 (0.42)	69.37 (5.47)	55.40 (2.26)	74.15 (11.78)	0.7736 (0.1253)	4.7084 (2.1537)	9.6586 (3.4559)	4.3581 (16.5047)
	Syn.	73.78 (0.51)	68.20 (5.17)	49.65 (1.00)	71.62 (10.90)	0.7938 (0.1237)	5.2044 (2.3468)	46.5505 (56.2730)	2.0651 (0.8180)

also be trained on and generalized to multiple subgraphs sampled from a large graph (in terms of large number of nodes). Most of existing GNNs that can be scaled up to large graphs merely focus on the (semi-)supervised node-level tasks (e.g., node classification) with available node attributes, while we consider the challenging *inductive* unsupervised node-level task (i.e., GP) across graphs without any attributes in this study. In our future work, we intend to further explore efficient (i) feature extraction and (ii) subgraph sampling strategies for IGP on large graphs without available graph attributes. In particular, we can first split a large graph into multiple small subgraphs via a sampling module and extract the neigh-induced features $\{\mathbf{X}_t\}$ for each sampled graph. The IGP framework can then be applied to multiple sampled graphs and derive the embedding $\{\mathbf{U}_t\}$ with respect to each subgraph. One possible challenge in this application is to ensure the consistency of embeddings $\{\mathbf{U}_t\}$ of subgraphs sampled from a common large graph (e.g., embeddings of all the subgraphs are mapped into a common latent space) without any auxiliary information given by the node attributes.