CGC: Contrastive Graph Clustering for Community Detection and Tracking

Namyong Park¹, Ryan Rossi², Eunyee Koh², Iftikhar Ahamath Burhanuddin², Sungchul Kim², Fan Du², Nesreen Ahmed³, Christos Faloutsos¹
{namyongp,christos}@cs.cmu.edu,{ryrossi,eunyee,burhanud,sukim,fdu}@adobe.com,nesreen.k.ahmed@intel.com
¹Carnegie Mellon University, ²Adobe Research, ³Intel Labs

ABSTRACT

Given entities and their interactions in the web data, which may have occurred at different time, how can we find communities of entities and track their evolution? In this paper, we approach this important task from graph clustering perspective. Recently, state-of-the-art clustering performance in various domains has been achieved by deep clustering methods. Especially, deep graph clustering (DGC) methods have successfully extended deep clustering to graph-structured data by learning node representations and cluster assignments in a joint optimization framework. Despite some differences in modeling choices (e.g., encoder architectures), existing DGC methods are mainly based on autoencoders and use the same clustering objective with relatively minor adaptations. Also, while many real-world graphs are dynamic, previous DGC methods considered only static graphs. In this work, we develop CGC, a novel end-to-end framework for graph clustering, which fundamentally differs from existing methods. CGC learns node embeddings and cluster assignments in a contrastive graph learning framework, where positive and negative samples are carefully selected in a multi-level scheme such that they reflect hierarchical community structures and network homophily. Also, we extend CGC for time-evolving data, where temporal graph clustering is performed in an incremental learning fashion, with the ability to detect change points. Extensive evaluation on real-world graphs demonstrates that the proposed CGC consistently outperforms existing methods.

CCS CONCEPTS
• Information systems → Clustering: Temporal data. Web mining. • Computing methodologies → Neural networks.

KEYWORDS
community detection and tracking, deep graph clustering, temporal graph clustering, contrastive learning, deep graph learning

1 INTRODUCTION

Given events between two entities, how can we effectively find communities of entities in an unsupervised manner? Also, when the events are associated with time, how can we detect communities and track their evolution? Various web platforms, including social networks, generate data that represent events between entities, occurring at a certain time, e.g., check-in records and user interaction logs. Finding communities from such dyadic temporal events can be formulated as a graph clustering problem, in which the goal is to find node clusters from a graph, where the two entities of an event are nodes, and the event forms a temporal edge between them.

In recent years, state-of-the-art clustering performance has been achieved by deep clustering methods in several application domains [20, 36, 37, 63–65, 67]. Following this success, deep graph clustering (DGC) [6, 40, 48, 57, 61] has been receiving increasing attention recently, which aims to learn cluster-friendly representations using deep neural networks for graph clustering. Early DGC methods [28, 57] have taken a two-stage approach, where representation learning and clustering are done in isolation; e.g., node embeddings are learned by graph autoencoders (GAEs) [28], to which a clustering method is applied. More accurate clustering results have been obtained by another group of DGC methods [6, 48, 61] that adopt a joint optimization framework, where a clustering objective is combined with the representation learning objective, and both are optimized simultaneously in an end-to-end manner.

In DGC methods, a major challenge lies in how to effectively utilize node features and graph structure. Graph neural networks provide an effective framework to this end, which propagate and aggregate node features over the graph, thus learning node embeddings that reflect network homophily. Further, to make the most of graph structure and node features, existing methods tried different modeling choices, e.g., in terms of encoder architectures (GAEs, attentional GAEs, GAEs with autoencoders (AEs)) and how graph structural features and node attributes are combined. Still, differences

| Desiderata         | Methods   |
|--------------------|-----------|
| CL                 | AE        |
| RP                 | GAE       |
| CL and RP          | DAERNN    |
| Input node features| DAEGC     |
| Network homophily  | SDCN      |
| Hierarchical       | AGCN      |
| Clustering         | CGC       |

Table 1: CGC wins on features. Comparison of the proposed CGC with deep learning approaches for graph clustering.

Historical features of CGC

[23] [28] [48] [57] [61] (Ours)

Jointly optimizing CL and RP ✓ ✓ ✓ ✓ ✓
[A] Input node features ✓ ✓ ✓ ✓ ✓ ✓
[A] Network homophily ✓ ✓ ✓ ✓ ✓ ✓
[A] Hierarchical communities ✓ ✓ ✓ ✓ ✓ ✓
Temporal graph clustering ✓ ✓ ✓ ✓ ✓ ✓

Learning Objective
Contrastive learning-based Reconstruction-based ✓ ✓ ✓ ✓ ✓ ✓ ✓
Figure 1: CGC outperforms competition: All points are below the diagonals for all baselines and graphs. CGC achieves more accurate (a) node clustering on static and temporal data, and (b) link prediction based on the time-evolving cluster membership.

among them are relative small: They mainly (1) perform reconstruction loss minimization for unsupervised representation learning (reconstructing the adjacency matrix, node attribute matrix, or both) in an AE-based framework, and (2) employ the clustering objective first proposed in DEC [63], which optimizes cluster assignments by learning from the model’s high confidence predictions.

In addition, while many real-world networks are dynamic in nature, no DGC methods are designed for clustering time-evolving graphs to our knowledge. Although we can apply existing methods to cluster temporal graphs (e.g., by ignoring time and applying them to the cumulative graph anew at each time step), practical solutions for temporal graph clustering should be able to incrementally learn changing community structures, and detect major change points, which cannot be addressed effectively by existing methods.

In this paper, we develop CGC, a new graph clustering framework based on contrastive learning, which significantly differs from existing DGC methods as summarized in Table 1. The main idea of contrastive learning [7, 27, 59] is to pull an entity (called an anchor) and its positive sample closer to each other in the embedding space, while pushing the anchor away from its negative sample. When no labels are available, the choice of positive and negative samples plays a crucial role in contrastive learning. In such cases, positive samples are often obtained by taking different views of the data (e.g., via data augmentations such as rotation and color distortion for images [7]), while negative samples are randomly selected from the entire pool of samples. In CGC, based on our understanding of real-world networks and their characteristics (e.g., homophily and hierarchical community structures), we design a multi-level scheme to choose positive and negative samples such that they reflect the underlying hierarchical communities and their semantics. Also, from information theoretic perspective, our contrastive learning objective is designed to maximize the mutual information between an entity and the hierarchical communities it belongs to in the latent space. Then guided by this multi-level contrastive objective, cluster memberships and entity embeddings are iteratively optimized in an end-to-end framework.

Furthermore, to find communities from time-evolving data, we extend CGC framework to the temporal graph clustering setting. Upon the arrival of new events, entity representations and cluster memberships are updated to reflect the new information, and at the same time, temporal smoothness assumption is incorporated into the GNN encoder, and also into the contrastive learning objective, which enables CGC to adapt to changing community structures in a controlled manner. We also show how CGC can be applied to detect major changes occurring in the network, and thereby adaptively choose homogeneous historical events to find communities from.

In summary, the key contributions of this work are as follows.

- **Novel Framework.** We propose CGC, a new contrastive graph clustering framework. As discussed above and summarized in Table 1, CGC is a significant departure from previous DGC methods.

- **Temporal Graph Clustering.** We extend our CGC framework for temporal data. CGC is the first deep graph clustering method for clustering time-evolving networks.

- **Effectiveness.** We demonstrate the effectiveness of CGC via extensive evaluation of clustering quality on several static and temporal real-world datasets (Figure 1).

### 2 PROBLEM FORMULATION

In this section, we introduce notations and definitions, and present the problem formulation. Table 2 lists the symbols used in this work.

#### 2.1 Graph Clustering

Let $G = (V, E)$ be a graph with nodes $V = \{1, \ldots, n\}$ and edges $E = \{(u_i, v_j) \mid u_i, v_j \in V\}_{i=1}^{m}$. Let $F \in \mathbb{R}^{n \times d}$ be an input node feature matrix. Let $k$ denote the number of node clusters. We define cluster membership as follows to represent node-to-cluster assignment.

**Definition 1 (Cluster Membership).** A cluster membership $\Phi_u \in \mathbb{R}^{k}_{\geq 0}$ of node $u$ is a stochastic vector that adds up to one, where the $i$-th entry is the probability of node $u$ belonging to $i$-th cluster.

According to Definition 1, a node belongs to at least one cluster, and can belong to multiple clusters. Note that this soft cluster membership includes hard cluster assignments as a special case, in which one node belongs to exactly one cluster. Based on this definition, graph clustering problem is formally defined as follows.

**Problem 1 (Graph Clustering).** Given a graph $G = (V, E)$ and input node features $F \in \mathbb{R}^{n \times d}$, learn a cluster membership matrix $\Phi \in \mathbb{R}^{n \times k}_{\geq 0}$ for all $n$ nodes in $G$.

After graph clustering, we want the nodes to be grouped such that nodes are more similar to those in the same cluster (e.g., in terms of external node labels if available, or connectivity patterns, node features, and structural roles) than nodes in different clusters.

#### 2.2 Temporal Graph Clustering

Let $G_t = (V, E_t)$ be a temporal graph snapshot with nodes $V = \{1, \ldots, n\}$ and temporal edges $E_t = \{(u, v, t) \mid u, v \in V, t \in \tau\}$, where $t$ is time (e.g., a timestamp at the level of milliseconds), and $\tau$ denotes some time span (e.g., one minute, one hour).
Definition 2 (Temporal Graph Stream). A temporal graph stream \( \mathcal{G} \) is a sequence of graph snapshots \( \mathcal{G} = \{ G_t \}_{t=1}^T \), where \( T \) is the number of graph snapshots thus far in the stream. Graph snapshots \( \{ G_t \} \) are assumed to be non-overlapping and ordered in increasing order of time.

Problem 2 (Temporal Graph Clustering). Given a temporal graph stream \( \mathcal{G} = \{ G_t \}_{t=1}^T \) and input node features \( F \in \mathbb{R}^{n \times d} \), learn a cluster membership matrix \( \Phi_t \in \mathbb{R}^{n \times k} \) for each time span \( t \).

3 PRELIMINARIES

Mutual Information (MI) and Contrastive Learning. The MI between two random variables (RVs) measures the amount of information obtained about one RV by observing the other RV. Formally, the MI between two RVs \( X \) and \( Y \), denoted \( I(X; Y) \), is defined as

\[
I(X; Y) = \mathbb{E}(\log \frac{p(x,y)}{p(x)p(y)})
\]

where \( p(x,y) \) is the joint density of \( X \) and \( Y \), and \( p(x) \) and \( p(y) \) denote the marginal densities of \( X \) and \( Y \), respectively. Several recent studies \([4, 7, 24, 59, 60]\) have seen successful results in representation learning by maximizing the MI between a learned representation and different aspects of the data.

Since it is difficult to directly estimate MI \([49]\), MI maximization is normally done by deriving a lower bound on MI and maximizing it instead. Intuitively, several lower bounds on MI are based on the idea that RVs \( X \) and \( Y \) have a high MI if samples drawn from their joint density \( p(x,y) \) and those drawn from the product of marginals \( p(x)p(y) \) can be distinguished accurately. InfoNCE \([59]\) is one such lower bound of MI in the form of a noise contrastive estimator \([21]\):

\[
I(X; Y) \geq \mathbb{E} \left[ \frac{1}{K} \sum_{i=1}^{K} \log \frac{\exp(f(x_i, y_i))}{\sum_{j=1}^{K} \exp(f(x_i, y_j))} \right] \triangleq I_{\text{NCE}}(X; Y)
\]

where the expectation is over \( K \) independent samples \( \{ x_i, y_i \}_{i=1}^{K} \) from the joint density \( p(x, y) \). Given a set of \( K \) independent samples, the critic function \( f(\cdot) \) aims to predict for each \( x_i \) which one of the \( K \) samples \( x_i \) was drawn together with, i.e., by assigning a large score to the positive pair \( (x_i, y_i) \), and small scores to other negative pairs \( \{ (x_i, y_j) \}_{j \neq i}^{K} \).

Graph Neural Networks (GNNs). GNNs are a class of deep learning architectures for graphs that produce node embeddings by repeatedly aggregating local node neighborhoods. In general, a GNN encoder \( \mathcal{E} \) maps a graph \( G \) and input node features \( F \in \mathbb{R}^{n \times d} \) into node embeddings \( H \in \mathbb{R}^{n \times d} \), that is, \( \mathcal{E}(G, F) = H \).

4 PROPOSED FRAMEWORK

In this section, we present the CGC framework. We describe how CGC performs graph clustering in a multi-level contrastive learning framework (Section 4.1), and discuss how we extend CGC for temporal graph clustering (Section 4.2).

4.1 CGC: Contrastive Graph Clustering

The proposed framework CGC performs contrastive graph clustering by carrying out the following two steps in an alternating fashion: (1) refining cluster memberships based on the current node embeddings, and (2) optimizing node embeddings such that nodes from the same cluster are closer to each other, while those from different clusters are pushed further away from each other.

4.1.1 Multi-Level Contrastive Learning Objective. In CGC, contrastive learning happens in the second step above, where positive samples of a node are assumed to have been generated by the same cluster as the node of interest, whereas negative samples are assumed to belong to different clusters. While no cluster membership labels are available, there exist several signals at different levels of the input data that we can utilize to effectively construct positive and negative samples for contrastive graph clustering, namely, input node features and the characteristics of real-world networks, such as network homophily and hierarchical community structure.

Signal: Input Node Features. Entities in the same community tend to have similar attributes. Thus informative node features can be used to distinguish nodes in the same class from those in different classes. Node features are especially helpful for sparse graphs, since they can complement the scarce relational information.

Therefore, for node \( u \), we take its input features \( F_u \) as its positive sample, and randomly select another node \( v \) to take its input features \( F_v \) as a negative sample; these positive and negative samples are then contrasted with node embedding \( h_u \). Let \( S_u = \{ F_i \}_{i=0}^n \) be the set of one positive \((i = 0)\) and \( r \) negative \((1 \leq i \leq r)\) samples \( (i.e., \text{input features}) \) for node \( u \), where \( r \) indicates that sampling was involved. Since input features and latent embeddings can have different dimensionality, we define a node feature-based contrastive loss \( L_F \) using a bilinear critic parameterized by \( W_F \in \mathbb{R}^{d \times d} \),

\[
L_F = \sum_{i=1}^{n} -\log \frac{\exp((h_u^i W_F F_u^i) / \tau)}{\sum_{i=0}^{r} \exp((h_u^i W_F F_u^i) / \tau)}
\]

where \( \tau > 0 \) is a temperature hyper-parameter.

Signal: Network Homophily. In real-world graphs, similar nodes are more likely to attach to each other than dissimilar ones, and accordingly, a node is more likely to belong to the same cluster as its neighbors than randomly chosen nodes. In particular, many real-world networks demonstrate the phenomenon of higher-order label homogeneity, i.e., the tendency of nodes participating in higher-order structures \( (e.g., \text{triangles}) \) to share the same label, which is a stronger signal than being connected by an edge alone. Thus, we use edges and triangles in constructing positive samples.
Further, CGC encodes nodes using GNNs, whose neighborhood aggregation scheme also enforces an inductive bias for network homophily that neighboring nodes have similar representations.

Let $N(u)$ denote the neighbors of node $u$. Let $N_\delta(u)$ be node $u$’s neighbors that participate in the same triangle as node $u$; thus, $N_\delta(u) \subseteq N(u)$. A positive sample for node $u$ is then chosen from among $N_\delta(u)$, with a probability of $\delta/|N_\delta(u)|$ for the neighbor in $N_\delta(u)$, and a probability of $(1-\delta)/|N(u)\setminus N_\delta(u)|$ for its other neighbors, where $\delta > 0$ determines the weight for nodes in $N_\delta(u)$. Then the positive sample’s embeddings are taken from $H=E(G, F)$.

To construct negative samples, we design a network corruption function $C(G, F)$, which constructs a negative network from the original graph $G$ and input node features $F$. Specifically, we define $C(\cdot)$ to return corrupted node features $\tilde{F}$, via row-wise shuffling of $F$, while preserving the graph $G$, i.e., $C(G, F) = (G, \tilde{F})$, which can be considered as randomly relocating nodes over the graph while maintaining the graph structure. Then negative node embeddings $\tilde{H} \in \mathbb{R}^{n \times d'}$ are obtained by applying the GNN encoder to $G$ and $\tilde{F}$, and $r$ negative samples and their embeddings are randomly chosen.

Let $S_u^\ell = \{h_u^{(l)}\}_{l=0}^{L}$ be the set containing the embeddings of one positive ($i = 0$) and $r$ negative ($1 \leq i \leq r$) samples for node $u$. In CGC, a homophily-based contrastive loss $L_H$ is defined as:

$$L_H = \sum_{u=1}^{n} -\log \frac{\exp(h_u \cdot h_u^0/\tau)}{\sum_{i=0}^{r} \exp(h_u \cdot h_u^i/\tau)}$$

where we use an inner product critic function with a temperature hyper-parameter $\tau > 0$, and $r$ denoting that sampling was involved.

**Signal: Hierarchical Community Structure.** The above loss terms contrast an entity with other individual entities and their input features, thereby learning community structure at a relatively low level. Here, we consider communities at a higher level than before by directly contrasting entities with communities.

CGC represents communities as a cluster centroid vector $c \in \mathbb{R}^{d'}$ in the same latent space as entities, so that the distance between an entity and cluster centroids reflects the entity’s degree of participation in different communities. To effectively optimize an entity embedding by contrasting it with communities, cluster centroids need to have been embedded such that they reflect the underlying community structures and the semantics of input node features. While the model’s initial embeddings of entities and clusters may not capture such community and semantic structures well, the above two objectives and the use of GNN encoders in CGC effectively guide the optimization process towards identifying meaningful cluster centroids, especially in the early stage of model training.

Importantly, real-world networks have been shown to exhibit hierarchical community structures. To model this phenomenon, we design CGC to group nodes into a varying number of clusters. For example, when we aim to group nodes into three clusters, we may also group the same set of nodes into ten and thirty clusters; then all clustering results taken together reveal hierarchical community structures in different levels of granularities.

Let $\mathcal{K} = \{k_t\}_{t=1}^L$ be the set of the number of clusters, and $C_t \in \mathbb{R}^{k_t \times d'}$ be the cluster centroid matrix for each $t$. Given the current node embeddings $H$ and cluster centroids $\{C_t\}_{t=1}^L$, positive samples for node $u$ are chosen to be the $L$ cluster centroids that node $u$ also group the same set of nodes into ten and thirty clusters; then

**Algorithm 1 ContrastiveGraphClustering**

**Input:** graph $G$, input node features $F \in \mathbb{R}^{n \times d}$, clustering algorithm $\Pi$, number of clusters $\mathcal{K} = \{k_t\}_{t=1}^L$, refinement interval $R$

**Output:** cluster membership matrix $\Phi \in \mathbb{R}^{n \times k_t}$, node embedding matrix $H \in \mathbb{R}^{n \times d'}$, cluster centroid matrix $C \in \mathbb{R}^{k_t \times d'}$

1. while not maxEpoch and not converged do
   2. $H = E(G, F)$ \(\text{\textcircled{Eq. (7)}}\)
   3. if epoch $\% R = 0$ then \(\text{\textcircled{refine clusters and memberships}}\)
   4. for $t = 1 \to L$ do
   5. $C_t, \Phi_t = \Pi(H, k_t)$
   6. Calculate loss $L$ using $H, F, \{C_t\}$ \(\text{\textcircled{Eqs. (3) to (6)}}\)
   7. Backpropagate and optimize model parameters
   8. $H = E(G, F)$
   9. $C, \Phi = \Pi(H, k_1)$
10. return $\Phi, H, C$

most strongly belongs to, while its negative samples are randomly selected from among the other $k_t - 1$ cluster centroids for each $t$. Let $S_u^\ell = \{h_u^{(l)}\}_{l=0}^{L}$ be the set with the embeddings of one positive ($i = 0$) and $r_t$ negative ($1 \leq i \leq r_t$) samples (i.e., centroids) for node $u$ chosen among $k_t$ centroids. Using an inner product critic, CGC defines a hierarchical community-based contrastive loss $L_C$ to be:

$$L_C = \sum_{u=1}^{n} -\frac{1}{L} \sum_{l=1}^{L} \log \frac{\exp(h_u \cdot c_u^o/\tau)}{\sum_{l=1}^{L} \exp(h_u \cdot c_u^l/\tau)}$$

**Multi-Level Contrastive Learning Objective.** The above loss terms capture signals on the community structure at multiple levels, i.e., individual node features ($L_F$), neighboring nodes ($L_H$), and hierarchically structured communities ($L_C$). CGC jointly optimizes

$$L = \lambda_F L_F + \lambda_H L_H + \lambda_C L_C$$

where $\lambda_F$, $\lambda_H$, and $\lambda_C$ are weights for the loss terms. Via multi-level contrastive estimation, CGC maximizes the MI between nodes and the communities they belong to in the learned latent space.

4.1.2 Encoder Architecture. As our node encoder $E$, we use a GNN with a mean aggregator.

$$h_u = \text{ReLU}(W_G \cdot \text{MEAN}(\{h_v^{\ell-1} \cup h_u^{\ell-1} \mid \forall u \in N(v)\}))$$

(7)

where node $v$’s embedding $h_v^{\ell}$ from the $l$-th layer of $G$ is obtained by averaging the embeddings of node $v$ and its neighbors from the $(l-1)$-th layer, followed by a linear transformation $W_G$ and ReLU non-linearity; $h_u^{\ell}$ is initialized to be the input node features $f_u$.

4.1.3 Algorithm. Algorithm 1 shows how (1) cluster memberships and (2) node embeddings are alternately optimized in CGC. (1) Given the current node embeddings $H$ produced by $E$ (line 2), a clustering algorithm $\Pi$ (e.g., $k$-means) refines cluster centroids $\{C_t\}$ and memberships $\{\Phi_t\}$ (lines 3-5). (2) Based on the updated cluster centroids and memberships, CGC computes the loss and optimizes model parameters (lines 6-7). In $\{k_t\}$, we assume that $k_1$ is the number of clusters that we ultimately want to identify in the network.

4.2 CGC for Temporal Graph Clustering

As a new graph snapshot $G_t$ arrives in a temporal graph stream $G = \{G_1, \ldots, G_{T+1}\}$, node embeddings $H_{t-1}$ and cluster memberships $\Phi_{t-1}$ that CGC learned from the snapshots until $(i-1)$-th time span are incrementally updated to reflect the new information.
in $G_t$. Specifically, given a sequence of graph snapshots, CGC merges them into a temporal graph and performs contrastive graph clustering, taking the temporal information into account. We use the notation $G_{t:j}$ to denote a temporal graph that merges the snapshots \( \{G_{t}, \ldots, G_{t:j}\} \), i.e., $G_{t:j} = (V,E_{t:j})$ where $E_{t:j} = \bigcup_{i=t}^{j} E_{ti}$. Below we describe how we extend CGC for temporal graph clustering.

4.2.1 Temporal Contrastive Learning Objective. As entities interact with each other, their characteristics may change over time, and such temporal changes normally occur smoothly. Thus, edges of a node observed across a range of time spans provide similar and related temporal views of the node in terms of its connectivity pattern. Accordingly, given node $u$ for time span $j$, we take its embedding $h_{u,j-1}$ obtained in the previous, $(j-1)$-th time span as its positive sample. To obtain negative samples, we use the same network corruption function used in Section 4.1.1, obtaining corrupted node embeddings $\tilde{F}$, and take node $u$’s embedding from the corrupted node embeddings $E(G_{t:j-1}, \tilde{F})$ as the negative sample; multiple negative samples can be obtained by using multiple sets of corrupted node features. Let $S^j_{t:j-1} = \{h_{u,j-1}(1), \ldots, h_{u,j-1}(n)\}$ be the set with the embeddings of one positive (i.e., $i=0$) and $r$ negative (i.e., $1 \leq r \leq t$) samples of node $u$ for the $j$-th time span, again $r$ denoting the involvement of sampling. CGC defines a time-based contrastive loss $L_T$ for time span $j$ to be:

$$L_T = \frac{1}{n} \sum_{i=1}^{n} \log \frac{\exp(h_{u,j} \cdot h_{u,j-1}/\tau)}{\sum_{i=0}^{n} \exp(h_{u,j} \cdot h_{u,j-1}/\tau)}$$

Note that Equation (8) is combined with the objectives discussed in Section 4.1.1 with a weight of $\lambda_T$, augmenting the loss $L$ to be

$$L = \lambda_F L_F + \lambda_H L_H + \lambda_C L_C + \lambda_T L_T.$$  

4.2.2 Encoder Architecture. We extend the GNN encoder such that in Section 4.1.1 with a weight of $\lambda$, denoting the involvement of sampling. CGC merges it with the previous ones, and refines cluster memberships on the resulting temporal graph. This process is based on the assumption that new events are similar to earlier ones. However, the new snapshot may differ greatly from the previous ones, when significant changes have occurred in the network. Detecting such changes is important, as it lets CGC find clusters from snapshots with similar patterns, and such events also correspond to important milestones or anomalies in the network.

Let $G_{seg} = \{G_t, \ldots, G_{t:j}\}$ be the current graph stream segment for some $j$ and $i$ ($j < i$). Given a new snapshot $G_{t:j+1}$, we expand the current segment $G_{seg}$ with $G_{t:j+1}$ if $G_{t:j+1}$ is similar to $G_{seg}$; if not, we start a new graph stream segment consisting only of $G_{t:j+1}$. This is basically a binary decision problem on whether to segment the graph stream or not. Our idea to solve this problem is to compare the embeddings of the nodes appearing in both $G_{seg}$ and $G_{t:j+1}$. Note that the GNN encoder in this step was trained with the graphs in the observed segment, and no further training has been performed on the new snapshot. Since embeddings from GNNs reflect the characteristics of nodes that CGC learned from the existing segment, the embeddings of the nodes in the new graph $G_{t:j+1}$ will be similar to their embeddings in the existing segment $G_{seg}$ if $G_{t:j+1}$ is similar to $G_{seg}$. By the same token, a major change in the new snapshot will lead to a large difference between the embeddings of a node in $G_{t:j+1}$ and $G_{seg}$. Let $V^*$ be the nodes appearing in both $G_{seg}$ and $G_{t:j+1}$. Let $H_{seg}^{V^*}, H_{t:j+1}^{V^*} \in \mathbb{R}^{n \times d'}$ be the two sets of embeddings of the nodes in $V^*$, computed for $G_{seg}$ and $G_{t:j+1}$, respectively, as discussed above. Using a distance metric $d(\cdot, \cdot)$ (e.g., cosine distance), we define the distance $\text{Dist}(\cdot, \cdot)$ between $H_{seg}^{V^*}$ and $H_{t:j+1}^{V^*}$ to be

$$\text{Dist}(H_{seg}^{V^*}, H_{t:j+1}^{V^*}) = \text{MEAN}(d((H_{seg}^{V^*}), (H_{t:j+1}^{V^*})) | i \in V^*)$$

and segment the stream if the distance is beyond a threshold (Alg. 3).

4.2.4 Putting Things Together. CGC tracks changing cluster memberships in an incremental end-to-end framework (Alg. 2). As a new graph snapshot arrives, CGC adaptively determines a sequence of graph snapshots to find clusters from, using Alg. 3 (line 3), and updates clustering results and node embeddings, using Alg. 1 (line 4).

5 EXPERIMENTS

The experiments are designed to answer the following questions:

- **RQ1 (Node Clustering):** Given static and temporal graphs, how accurately can the proposed CGC cluster nodes? (Section 5.3)
- **RQ2 (Temporal Link Prediction):** How informative is the learned cluster membership in predicting temporal links? (Section 5.4)
- **RQ3 (Ablation Study):** How do different variants of the proposed framework affect the clustering performance? (Section 5.5)

Further results are in Appendix, e.g., mining case studies (App. A).

5.1 Datasets

5.1.1 Static Datasets. Table 7 presents the statistics of static datasets. These datasets have labels and input features for all nodes. ACM is a paper network from the ACM digital library [33], where two papers are linked by an edge if they are written by the same author. Papers in this dataset are published in KDD, SIGMOD, SIGCOMM, and MobiCom, and belong to one of the following three classes: database, wireless communication, and data mining. Node features are the bag-of-words of the paper keywords. DBLP-S is an author network from the DBLP computer science bibliography [11], where an edge connects two authors (i.e., nodes).
if they have a coauthor relationship. Authors are divided into the following four areas, according to the conferences of their publications: database, data mining, machine learning, and information retrieval. Node features are the bag-of-words of their keywords.

**Citeseer** is a citation network from the CiteSeer digital library [9], where an edge represents a citation between two documents. Documents are assigned to one of the six areas: agents, AI, database, information retrieval, machine learning, and human-computer interaction. Node features are the bag-of-words of the documents.

**MAG-CS** is a network of authors in CS from the Microsoft Academic Graph. An edge connects two authors (i.e., nodes) if they co-authored a paper. Node features are keywords of the author’s papers, and node labels denote most active field of study of each author.

**DBLP-T** is an author network from DBLP [11], where edges denote coauthorship from 2004 to 2018. Node labels represent the authors’ research areas (computer networks and machine learning), and may change over time as authors switch their research focus.

**Yahoo-Msg** is a communication network among Yahoo! Messenger users [50], where two users are linked by an edge if a user sent a message to another user.

**Foursquare-NYC** and **Foursquare-TKY** are user check-in records, collected by Foursquare [12] between April 2012 and February 2013 from New York City and Tokyo, respectively. An edge links a user and a venue if a user checked in to the venue.

### 5.2 Baselines

**Static Baselines.** K-means [22] is a classic clustering method applied to the raw input features. AE [23] produces node embeddings by using autoencoders. DEC [63] is a deep clustering method that optimizes node embeddings and performs clustering simultaneously. IDEC [20] extends DEC by adding a reconstruction loss. A group of methods also take graph structures into account for node representation learning and graph clustering. SVD [15] applies singular value decomposition to the adjacency matrix. GAE [28] and VGAE [28] employ a graph autoencoder and a variational variant. ARGa [40] and ARGaVA [40] are an adversarially regularized graph autoencoder and its variational version. DGI [60] learns node embeddings by maximizing their MI with the graph. DAEGC [61], SDCN [6], and AGCN [48] are deep graph clustering methods that jointly optimize node embeddings and graph clustering.

**Temporal Baselines.** CTDE [38] learns node embeddings based on temporal random walks. TIMERS [68] is an incremental SVD method that employs error-bounded SVD restart on dynamic networks. DynGEM [17] leverages AEs to incrementally generate node embeddings at time $t$ by using the graph snapshot at time $t-1$. DynAERNN [16] uses historical adjacency matrices to reconstruct the current one by using an encoder-decoder architecture with RNNs. EvolveGCN [41] models how the parameters of GCNs [29] evolve over time. CTGCN [35] is a $k$-core based temporal GCN.

For methods that produce only node embeddings (e.g., AE, SVD, GAE, CTDE), we apply $k$-means to the node embeddings to obtain cluster memberships. As the temporal link prediction task in Section 5.4 involves dot product scores, we apply Gaussian mixture models to node embeddings to obtain soft cluster memberships. Appendix C presents experimental settings of baselines and CGC.

### 5.3 Node Clustering Quality (RQ1)

We evaluate the clustering quality using static and temporal graphs with node labels (Citeseer, DBLP-S, ACM, MAG-CS, and DBLP-T). Given cluster assignments, the best match between clusters and node labels is obtained by the Munkres algorithm [30], and clustering performance is measured using four metrics, which range from 0 to 1 (higher values are better): ACC (Accuracy), NMI (Normalized Mutual Information), ARI (Adjusted Rand Index), and F1 score.

#### 5.3.1 Static Datasets.** Table 3 shows the results on static graphs. The proposed method CGC consistently outperforms existing methods on all datasets in four metrics. Our novel multi-level contrastive graph learning objectives enable CGC to accurately identify node clusters by effectively leveraging the characteristics of real-world networks. We summarize our observations on the results below.

1. Deep clustering methods (DEC, IDEC) outperform AE, which performs dimensionality reduction of the input features without clustering objectives.
2. Comparing AE against GAE and ARGA, we can see that utilizing graph structures improves the clustering quality; in some cases, the performance of GAE and ARGA is even better than DEC and IDEC, although they do not have clustering objectives.
3. Deep graph clustering methods (DAEGC, SDCN, AGCN) further improve upon deep clustering methods and those that learn from input features or the graph structure without clustering objectives, which shows the benefit of combining deep clustering with graph structural information.
4. A comparison with DGI is also noteworthy, as DGI learns node embeddings via MI maximization over a graph. Despite some similarity, DGI cannot effectively identify community structures, as it maximizes the MI between nodes and the entire graph, without regard to communities therein.

#### 5.3.2 Temporal Datasets.** Results on the temporal graph DBLP-T are in Table 4, which reports the average of the clustering performance over multiple temporal snapshots. Since static baselines have no notion of graph stream segmentation, it is up to the user to decide which data to provide as input. We evaluate static baselines in two widely used settings, representative of the way existing temporal graph clustering methods operate: The default setting is to use all observed snapshots at each time step, and the other setting is to use only the latest graph snapshot (marked with “-latest” suffix).

CGC outperforms all baselines, achieving up to 13% and 397% higher ACC and NMI, respectively, than the best performing baseline. Notably, nearly all baselines do not perform well, obtaining close to zero NMI and ARI, which demonstrates the difficulty of finding clusters over time-evolving networks. Especially, no input features are available for DBLP-T, which poses an additional challenge to methods that heavily rely on them. For static baselines, using all snapshots often led to similar or better results in comparison to using the last snapshot. Results also show that temporal baselines fail to identify changing community structure. While they are designed to keep track of time-evolving node embeddings, their representation learning mechanism does not take clustering objective into account, which makes them less effective for community detection. Figure 6a in Appendix B shows how ACC and NMI of CGC and four select baselines change over time. While baselines’ performance shows an upward trend, their improvement is not significant. On the other hand, CGC’s performance improves remarkably over time, successfully identifying changing communities.
5.4 Temporal Link Prediction Accuracy (RQ2)

The task is to predict the graph $G_T = (V, E_T)$ for the next time span $\tau'$, where $E_T$ are the temporal positive (i.e., observed) edges. We uniformly randomly sample the same amount of temporal negative edges $E_{\neg T}$ such that $E_{\neg T} = \{(u, v) | u, v \sim \text{Uniform}(1, \ldots , n) \land (u, v) \notin E_T\}$. Given an edge $\langle u, v \rangle \in E_T \cup E_{\neg T}$ for time span $\tau$ to predict, we estimate the likelihood of such an edge existing as $A_U^{\tau} = \phi_u^T \phi_v$, where $\phi_u$ and $\phi_v$ are cluster memberships for nodes $u$ and $v$. We can use link prediction task for evaluating clustering quality, since the temporal link prediction is based on the time-evolving membership vector $\phi$, it summarizes how accurately the learned cluster memberships capture temporally-evolving community structure. Table 5 reports the link prediction accuracy in terms of the area under the receiver operating characteristic curve (AUC) and the average precision (AP). Both metrics range from 0 to 1, and higher values are better. As the number of test edges (i.e., $E_T \cup E_{\neg T}$) changes over time, we average the performance for each snapshot, weighted by the size of $E_T \cup E_{\neg T}$. Results show that CGC consistently outperforms baselines on all datasets, achieving up to 29% higher temporal link prediction performance. The best results among baselines were mainly obtained by CTGCN, which is a temporal method that models the network evolution. Among static baselines, AGCN mostly outperforms other static methods, and even most dynamic baselines, except CTGCN. This can be explained by the fact that these dynamic baselines are trained using cluster agnostic objectives, which again shows that incorporating the clustering objective can be helpful for detecting communities. As in Section 5.3.2, we report results obtained in the two settings (i.e., all vs. latest) for static baselines. There is no clear winner between them. Figure 6b shows how the performance of CGC and four baselines changes over time.

5.5 Ablation Study (RQ3)

We investigate how contrasting learning objectives affects CGC. Figure 2 shows node clustering results where CGC was trained with different combinations of contrastive objectives; $F$, $H$, and $C$ denote the loss terms on node features ($\ell_F$), network homophily ($\ell_H$), and hierarchical communities ($\ell_C$) in Eq. (6), respectively, and only the specified terms were included with a weight of 1. We report relative scores, i.e., scores divided by the best score for each metric. Results show that the proposed contrastive objectives are complementary, i.e., jointly optimizing these objectives improves the performance, e.g., $F$ to $F$ on ACM and $H$ to $H$ on DBLP-T. Especially, the best result on ACM and DBLP-T is obtained when all objectives are used together ($F$+$H$+$C$). However, DBLP-T shows a different pattern, where the best result was obtained with $F$+$C$. Notably, in DBLP-T, the objective on network homophily was not useful whether it is used alone (H) or with others ($F$+$F$+$H$). In DBLP-T, 36% of the nodes are isolated, making it hard to learn from graph structure. Still, joint optimization improved the results (e.g., $H$ vs. $H$+$C$).

6 RELATED WORK

Graph Clustering. Several approaches have been developed or adapted for semi-supervised or structured data clustering, including...
Table 5: CGC consistently outperforms baselines, achieving up to 29% higher temporal link prediction performance than the best baseline. Best results are in bold, and second best results are underlined.

| Method         | Foursquare-NYC | Foursquare-TKY | Yahoo-Msg |
|----------------|----------------|----------------|-----------|
|                | ROC AUC | Avg. Prec. | ROC AUC | Avg. Prec. | ROC AUC | Avg. Prec. |
| SVD [15]       | 9.68±0.3 | 33.28±0.2 | 4.18±0.0 | 37.84±0.0 | 59.51±0.5 | 64.88±0.4 |
| SVD-latest     | 17.67±0.6 | 37.93±0.6 | 7.20±0.2 | 35.08±0.1 | 49.26±0.2 | 53.21±0.2 |
| DGI [60]       | 14.37±0.7 | 33.02±0.1 | 13.79±1.0 | 33.17±0.3 | 50.60±0.5 | 51.83±0.3 |
| DGI-latest     | 18.55±1.2 | 34.16±0.3 | 20.01±0.7 | 34.69±0.3 | 41.92±0.2 | 45.00±0.2 |
| GAE [28]       | 13.55±1.1 | 33.16±0.3 | 17.44±0.5 | 35.01±0.3 | 46.40±0.5 | 48.41±0.2 |
| GAE-latest     | 19.80±0.4 | 35.13±0.3 | 21.67±0.7 | 37.44±0.5 | 42.45±0.5 | 44.87±0.2 |
| VGEA [28]      | 6.63±0.1 | 32.34±0.1 | 10.06±0.3 | 34.90±0.4 | 39.97±0.0 | 47.99±0.1 |
| VGEA-latest    | 12.02±0.2 | 33.18±0.0 | 12.91±0.2 | 34.93±0.2 | 44.21±0.1 | 49.61±0.0 |
| ARGA [40]      | 6.96±0.0 | 31.63±0.0 | 11.45±0.1 | 33.00±0.3 | 38.79±0.1 | 44.17±0.1 |
| ARGA-latest    | 11.89±1.1 | 32.38±0.2 | 13.17±0.2 | 32.61±0.0 | 39.84±0.1 | 43.78±0.0 |
| ARGA-VGEA [40] | 13.56±0.4 | 34.95±0.2 | 22.30±0.4 | 43.11±0.3 | 46.99±0.1 | 50.44±0.1 |
| ARGVA-latest   | 20.01±0.7 | 39.11±0.3 | 32.01±0.5 | 45.14±0.1 | 50.54±0.1 | 51.25±0.1 |
| SDCN [6]       | 47.86±0.7 | 46.31±0.6 | 37.32±0.8 | 40.73±0.6 | 55.76±1.5 | 55.78±1.3 |
| SDCN-latest    | 25.24±0.3 | 36.47±0.3 | 19.01±1.3 | 35.05±0.9 | 54.51±0.6 | 55.35±0.5 |
| AGCN [48]      | 56.13±1.0 | 52.24±1.5 | 42.43±2.7 | 42.44±2.2 | 54.23±2.2 | 54.43±1.5 |
| AGCN-latest    | 56.14±2.3 | 52.41±2.5 | 41.44±5.8 | 51.27±4.0 | 51.81±1.1 | 52.87±0.4 |

CGC (Ours) 64.60±0.6 70.34±0.5 66.26±0.8 70.22±0.6 84.30±0.1 86.88±0.1

Figure 2: Node clustering performance of CGC, obtained with different contrastive objectives. F: Feature Nodes. H: Network Homophily. C: Hierarchical Communities.
[61] Chun Wang, Shirui Pan, Ruqi Hu, Guodong Long, Jing Jiang, and Chengqi Zhang. 2019. Attributed Graph Clustering: A Deep Attentional Embedding Approach. In IJCAI. ijcai.org, 3670–3676.

[62] Xiao Wang, Nian Liu, Hui Han, and Chuan Shi. 2021. Self-supervised Heterogeneous Graph Neural Network with Co-contrastive Learning. In KDD. ACM, 1726–1736.

[63] Junyu Xu, Ross B. Girshick, and Ali Farhadi. 2016. Unsupervised Deep Embedding for Clustering Analysis. In ICML (JMLR Workshop and Conference Proceedings, Vol. 48). JMLR.org, 478–487.

[64] Bo Yang, Xiao Fu, Nicholas D. Sidiropoulos, and Mingyi Hong. 2017. Towards K-means-friendly Spaces: Simultaneous Deep Learning and Clustering. In ICML (Proceedings of Machine Learning Research, Vol. 70). PMLR, 3861–3870.

[65] Di Yao, Chao Zhang, Zhihua Zhu, Jian-Hui Huang, and Jingping Bi. 2017. Trajectory clustering via deep representation learning. In IJCNN. IEEE, 3880–3887.

[66] Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shou. 2020. When Does Self-Supervision Help Graph Convolutional Networks?. In ICML (Proceedings of Machine Learning Research, Vol. 119). PMLR, 10871–10880.

[67] Mingxuan Yue, Yaguang Li, Hauze Yang, Ritesh Ahuja, Yao-Yi Chiang, and Cyrus Shahabi. 2019. DETECT: Deep Trajectory Clustering for Mobility-Behavior Analysis. In IEEE BigData. IEEE, 988–997.

[68] Ziwei Zhang, Peng Cui, Jian Pei, Xiao Wang, and Wenwu Zhu. 2018. TIMERS: Error-Bounded SVD Restart on Dynamic Networks. In AAAI. AAAI Press, 224–231.
A MINING CASE STUDIES

A.1 Case Studies on Synthetic Graphs

In this section, we show how effectively CGC performs community detection and tracking, using synthetic graphs that consist of a small number of groups; each group corresponds to a tightly knit community, which experiences significant changes over time.

Figure 3: Two groups with traveling members (Case 1). Segmentation reveals a clearer community structure across time.

Case 1: Two Groups With Traveling Members (Figure 3). We have groups 1 and 2 for time 0-2. At time 3, half of the nodes in group 1 move to group 2, stay there until time 5, and then at time 6, move back to group 1, where they originally belonged. Thus there are two change points (CPs), i.e., time 3 and 6 (Figure 3d). Figure 3a shows the segment prior to the first CP. Figures 3b and 3c show the segment at the first CP when the graph stream was properly segmented or not; Figure 3c does not clearly show the change in the size of two groups. By performing segmentation in the presence of a significant change, CGC captures a clearer community structure.

Figure 4: Two groups reorganizing into three (Case 2). CGC identifies reorganizing communities and detects the change point.

Case 2: Two Groups Reorganizing Into Three (Figure 4). This network initially consists of two communities, which are regrouped into three communities due to a major reorganization at time 3. Figure 4a shows the two communities captured by CGC before the CP at time 3. Figure 4b shows that CGC successfully detects the CP (Figure 4d), and discovers restructured communities. Again, when the CP is ignored, it gets harder to see a clear structure of three communities from the resulting graph stream segment (Figure 4c).

A.2 Case Studies on Real-World Graphs

To see how the cluster membership found by CGC evolves over time, we cluster nodes based on the transition pattern (TP) of their membership vectors (Figure 5 (top)). Specifically, we concatenate the cluster membership vectors of each node obtained at different time steps, apply t-SNE to embed nodes in a two-dimensional space, and perform k-means clustering on the resulting two-dimensional node embeddings to obtain TP clusters. Then for each TP, we consider how cluster distribution changed over time (Figure 5 (bottom)).

Figure 5: Node clusters (top) based on their transition patterns (bottom) in the Yahoo-Msg dataset.

For each time step, we take the average of the membership vectors of the nodes belonging to a specific TP, and display the cluster distribution at each time as a column; clusters are associated with distinct colors, and the cluster distribution in the averaged membership vector at different time is shown by the proportion of the corresponding colors.

Yahoo-Msg (Figure 5). Nodes are clustered into 10 TPs. Among them, TP 0 shows a different pattern than others, where a major cluster changes frequently over time (e.g., switches between orange and green). In the scatter plot above, TP 0 is the cluster at the center, located close to a few surrounding clusters. Over time, the cluster assignments of nearby clusters have had a varying impact on how the nodes in TP 0 are clustered. Also, note that a segmentation occurred at the second time step, as can be seen in the TP plots. The color distribution of the first column in the four TPs greatly differs from those of the second and subsequent columns. Via segmentation, CGC discovers a clearer community structure.

B CLUSTERING PERFORMANCE OVER TIME

Figure 6 shows how the performance of CGC and four select baselines changes over time. For static baselines, we report the results obtained by clustering all observed graph snapshots at each time step.

Node Clustering (Figure 6a). While all methods do not perform well for the first few time steps, CGC’s performance continuously improves over time, reaching an ACC of ~0.89 and an NMI of ~0.48 in the end. Although baselines’ performance also improves with time, their improvement is much smaller than that of CGC, failing to effectively track the evolution of communities in the network.

Link Prediction (Figure 6b). CGC significantly outperforms baselines throughout most of the time span. Dynamic methods are not effective at capturing community structure, while deep clustering baselines like AGCN fail to track the evolution of clusters.

C EXPERIMENTAL SETTINGS

Experiments for Static Data. For ACM, DBLP-S, and Citeseer, we cite the results of all baselines (except SVD, DGI and AGCN) from [6]. For AGCN, we take its result from [48]. Settings of these baselines are given in [6, 48]. We directly evaluate SVD and DGI on these datasets. On MAG-CS, we evaluate all baselines using the settings in [6, 48]. For methods we evaluate, we report results averaged over 5 runs. We set node embedding size to 200 for SVD [53] and
For both node clustering (Table 4) and link prediction (Table 5) evaluation, baselines used mostly the same settings. We set the size of initial node features and latent node embeddings to 128 and 32, respectively, and used the Adam optimizer with a learning rate of 0.001. Since the datasets used for temporal link prediction (Yahoo-Msg, Foursquare-NYC, Foursquare-TKY) do not have ground truth clusters, we set the size of cluster membership to 64 for all baselines and CGC. Tables 4 and 5 report results averaged over five runs.

For GDG and DGI, we used the same setting used for static graphs. For GAE, VGAE, ARGA, and ARGW, we used the implementation of the PyTorch Geometric [13] with two-layer GCN encoders. For SDNC and AGCN, we used the default settings used in [6, 48], while setting the size of node embeddings to 32. For CTDNE, we used the default settings of the open source implementation [54]. We set $\theta$ in TIMERS to 0.17. In DynGEM, we set $\alpha$ to $10^{-5}$, $\beta$ to 10, and both $v_1$ and $v_2$ to $10^{-4}$. For DynAERNN, we set $\beta$ to 5, the look back parameter to 3, and both $v_1$ and $v_2$ to $10^{-6}$. In EvolveGCN, we used a two-layer GCN; specifically, we used EvolveGCN-H, which incorporates node embeddings in RNNs. For CTGCN, we used the CTGCN-C version with the settings used in [35]. In CGC, we set $\lambda_H = 1$, $\lambda_r = \lambda_f = 0.2$, $\lambda_p = 0$, $\Psi = 0.99$, $\theta = 0.3$. Let $r_f$ be the number of negatives per positive sample for the loss $L_f$. For all temporal datasets, we set $r_f = \tau_f = r_f = 10$. We set $r_C$ to 60 (link prediction datasets) and 30 (DBLP-T) for each $f$. For ACM, Foursquare-TKY, we set $K = (5k, 25k)$ for DBLP-T, and $K = (k, 5k, 25k)$ for all others. For CGC, we set the learning rate to 0.005, and the node embedding size to 32.

## D GRAPH STREAM SEGMENTATION

Alg. 3 shows how CGC decides whether to segment the graph stream or not. A description of Alg. 3 is given in Sec. 4.2.3.

### Algorithm 3 GraphStreamSegmentation

**Input:** graph stream segment $G_{\text{seg}}$, new graph $G_{\tau_{j+1}}$, for time span $j+1$, input node features $F \in \mathbb{R}^{n \times d}$, segmentation threshold $\theta$

**Output:** graph stream segment $G_{\text{seg}}$

1. if $G_{\text{seg}} \neq \emptyset$ then
2. $G_{\text{seg}} = \text{Merge}(G_{\text{seg}})$
3. $V^* = \text{Nodes}(G_{\text{seg}}) \cap \text{Nodes}(G_{\tau_{j+1}})$
4. $\mathcal{H}^W = \mathcal{E}(G_{\text{seg}}, F)$
5. $H^{\text{H}_1} = \mathcal{E}(G_{\tau_{j+1}}, F)$
6. if $G_{\text{seg}} = \emptyset \lor \text{Dist}(\mathcal{H}^W, H^{\text{H}_1}) > \theta$ then
7. $G_{\text{seg}} = \{G_{\tau_{j+1}}\}$ \texttt{Start a new graph stream segment.}
8. else
9. $G_{\text{seg}} = G_{\text{seg}} \cup \{G_{\tau_{j+1}}\}$ \texttt{Add $G_{\tau_{j+1}}$ to the current segment.}
10. return $G_{\text{seg}}$

## E ADDITIONAL RELATED WORK

Deep Clustering (DC). PARTY [47] is a two-stage DC method that uses autoencoders (AEs) with sparsity prior. DEC [63] is a single-stage AE-based method that jointly learns latent embeddings and cluster assignments by minimizing the KL divergence between the model’s soft assignment and an auxiliary target distribution. IDEC [20] further improves DEC by integrating DEC’s clustering loss and AE’s reconstruction loss. DCN [64] adopts the K-means objective to help AEs learn K-means-friendly representations. In [25], variational AEs are used to model the data generative procedure for DC. Recently, adversarial fairness has also been incorporated for deep fair clustering [32].