CGC: Contrastive Graph Clustering for Community Detection and Tracking

Namyong Park[1], Ryan Rossi[2], Eunyee Koh[2], Iftikhar Ahamath Burhanuddin[2], Sungchul Kim[2], Fan Du[2], Nesreen Ahmed[3], Christos Faloutsos[1]
{namyongp,christos}@cs.cmu.edu,{ryrossi,eunyee,burhanud,sukim,fdu}@adobe.com,nesreen.k.ahmed@intel.com
[1]Carnegie Mellon University, [2]Adobe Research, [3]Intel 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

ACM Reference Format:
Namyong Park, Ryan Rossi, Eunyee Koh, Iftikhar Ahamath Burhanuddin, Sungchul Kim, Fan Du, Nesreen Ahmed, Christos Faloutsos. 2022. CGC: Contrastive Graph Clustering for Community Detection and Tracking. In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3485447.3512160

WWW '22, April 25–29, 2022, Virtual Event, Lyon, France.
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9096-5/22/04.
https://doi.org/10.1145/3485447.3512160

Table 1: CGC wins on features. Comparison of the proposed CGC with deep learning approaches for graph clustering. [A]: Aware of/Utilizing. CL: Clustering, RP: Representation.

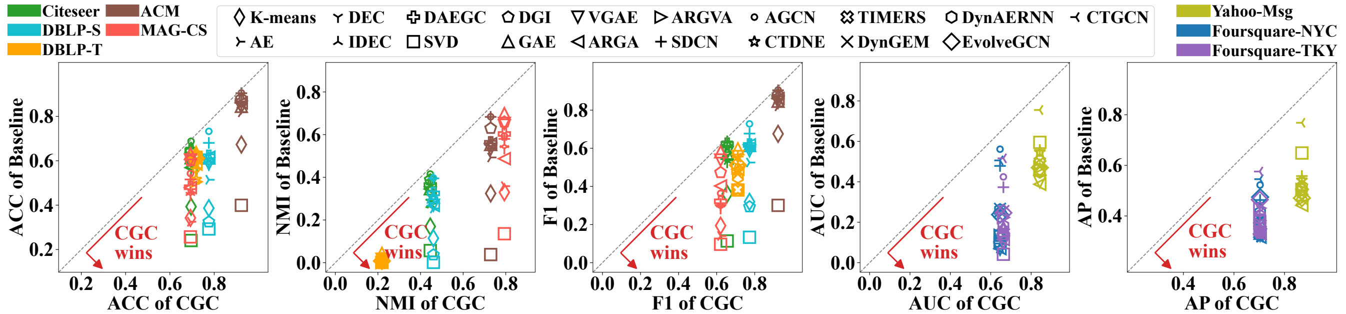
Methods *Desiderata*	AE [23]	GAE [28]	DAERNN [16]	DAEGC [61]	SDCN [6]	AGCN [48]	**CGC** **(Ours)**
Jointly optimizing CL and RP				✓	✓	✓	✓
[A] Input node features	✓	✓		✓	✓	✓	✓
[A] Network homophily		✓	✓	✓	✓	✓	✓
[A] Hierarchical communities							✓
Temporal graph clustering			✓				✓
Learning Objective							
Contrastive learning-based							■
Reconstruction-based	■	■	■	■	■	■	

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



(a) Node Clustering for Static and Temporal Datasets (b) Temporal Link Prediction

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 $E = \{(u_i, v_i) \mid u_i, v_i \in V\}_{i=1}^{m}$. Let $\mathbf{F} \in \mathbb{R}^{n \times d}$ 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 $\boldsymbol{\phi}_u \in \mathbb{R}_{\geq 0}^{k}$ 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 $\mathbf{F} \in \mathbb{R}^{n \times d}$, learn a cluster membership matrix $\boldsymbol{\Phi} \in \mathbb{R}_{\geq 0}^{n \times k}$ 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 $G_\tau = (V, E_\tau)$ be a temporal graph snapshot with nodes $V = \{1, \ldots, n\}$ and temporal edges $E_\tau = \{(u, v, t) \mid u, v \in V, \ t \in \tau\}$, where t is time (*e.g.*, a timestamp at the level of milliseconds), and τ 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 $\mathcal{G} = \{G_{\tau_i}\}_{i=1}^{T}$ where T is the number of graph snapshots thus far in the stream. Graph snapshots $\{G_{\tau_i}\}$ are assumed to be non-overlapping and ordered in increasing order of time.

Problem 2 (Temporal Graph Clustering). Given a temporal graph stream $\mathcal{G} = \{G_{\tau_i}\}_{i=1}^{T}$ and input node features $\mathbf{F} \in \mathbb{R}^{n \times d}$, learn a cluster membership matrix $\mathbf{\Phi}_i \in \mathbb{R}_{\geq 0}^{n \times k}$ for each time span τ_i.

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

$$I(X;Y) = \mathbb{E}_{p(x,y)}\left[\log(p(x,y)/p(x)p(y))\right] \tag{1}$$

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]:

$$I(X;Y) \geq \mathbb{E}\left[\frac{1}{K}\sum_{i=1}^{K} \log \frac{\exp(f(x_i, y_i))}{\frac{1}{K}\sum_{j=1}^{K}\exp(f(x_i, y_j))}\right] \triangleq I_{\text{NCE}}(X;Y) \tag{2}$$

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 $\mathbf{F} \in \mathbb{R}^{n \times d}$ into node embeddings $\mathbf{H} \in \mathbb{R}^{n \times d'}$, that is, $\mathcal{E}(G, \mathbf{F}) = \mathbf{H}$.

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.

Table 2: Table of symbols.

Symbol	Definition
u, v	node indices
n	number of nodes
k	number of clusters
t	timestamp of an edge, $t \geq 0$
τ	time span
$G = (V, E)$	static graph with nodes V and edges E
$\boldsymbol{\phi}_u \in \mathbb{R}_{\geq 0}^{k}$	cluster membership vector of node u for graph G
$G_\tau = (V, E_\tau)$	temporal graph snapshot with nodes V and temporal edges for time span τ
$\mathcal{G} = \{G_{\tau_i}\}$	temporal graph stream
$\mathbf{\Phi}_i \in \mathbb{R}_{\geq 0}^{n \times k}$	cluster membership matrix for time span τ_i
$\mathbf{F} \in \mathbb{R}^{n \times d}$	input node feature matrix
$\mathbf{H} \in \mathbb{R}^{n \times d'}$	node embedding matrix
$\mathcal{N}(u)$ $(\mathcal{N}_\Delta(u))$	neighbors of node u (participating in triangles with u)
$\mathcal{K} = \{k_\ell\}_{\ell=1}^{L}$	number of clusters for contrastive learning

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 $\mathbf{f}_u$ as its positive sample, and randomly select another node v to take its input features $\mathbf{f}_v$ as a negative sample; these positive and negative samples are then contrasted with node embedding $\mathbf{h}_u$. Let $\mathcal{S}_u^F = \{\mathbf{f}_u'^i\}_{i=0}^{r}$ be the set of one positive ($i = 0$) and r negative ($1 \leq i \leq r$) samples (*i.e.*, input features) for node u, where $'$ indicates that sampling was involved. Since input features and latent embeddings can have different dimensionality, we define a node feature-based contrastive loss $\mathcal{L}_F$ using a bilinear critic parameterized by $\mathbf{W}_F \in \mathbb{R}^{d' \times d}$:

$$\mathcal{L}_F = \sum_{u=1}^{n} -\log \frac{\exp((\mathbf{h}_u^\intercal \, \mathbf{W}_F \, \mathbf{f}_u'^0)/\tau)}{\sum_{v=0}^{r}\exp((\mathbf{h}_u^\intercal \, \mathbf{W}_F \, \mathbf{f}_u'^v)/\tau)} \tag{3}$$

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

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

Let $\mathcal{S}_u^H = \{\mathbf{h}_u'^i\}_{i=0}^r$ 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 $\mathcal{L}_H$ is defined as:

$$\mathcal{L}_H = \sum_{u=1}^{n} -\log \frac{\exp(\mathbf{h}_u \cdot \mathbf{h}_u'^0/\tau)}{\sum_{v=0}^{r} \exp(\mathbf{h}_u \cdot \mathbf{h}_u'^v/\tau)} \tag{4}$$

where we use an inner product critic function with a temperature hyper-parameter $\tau > 0$, and $'$ 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 $\mathbf{c} \in \mathbb{R}^{d'}$ 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.

Algorithm 1 ContrastiveGraphClustering

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

Output: cluster membership matrix $\Phi \in \mathbb{R}_{\geq 0}^{n \times k_1}$, node embedding matrix $\mathbf{H} \in \mathbb{R}^{n \times d'}$, cluster centroid matrix $\mathbf{C} \in \mathbb{R}^{k_1 \times d'}$

```
1  while not maxEpoch and not converged do
2      $\mathbf{H} = \mathcal{E}(G, \mathbf{F})$                              ▷Eq. (7)
3      if epoch % $R = 0$ then          ▷refine clusters and memberships
4          for $\ell = 1$ to $L$ do
5              $\mathbf{C}_\ell, \Phi_\ell = \Pi(\mathbf{H}, k_\ell)$
6      Calculate loss $\mathcal{L}$ using $\mathbf{H}, \mathbf{F}, \{\mathbf{C}_\ell\}$          ▷Eqs. (3) to (6)
7      Backpropagate and optimize model parameters
8  $\mathbf{H} = \mathcal{E}(G, \mathbf{F})$
9  $\mathbf{C}, \Phi = \Pi(\mathbf{H}, k_1)$
10 return $\Phi, \mathbf{H}, \mathbf{C}$
```

Let $\mathcal{K} = \{k_\ell\}_{\ell=1}^L$ be the set of the number of clusters, and $\mathbf{C}_\ell \in \mathbb{R}^{k_\ell \times d'}$ be the cluster centroid matrix for each ℓ. Given the current node embeddings $\mathbf{H}$ and cluster centroids $\{\mathbf{C}_\ell\}_{\ell=1}^L$, positive samples for node u are chosen to be the L cluster centroids that node u most strongly belongs to, while its negative samples are randomly selected from among the other $k_\ell - 1$ cluster centroids for each ℓ. Let $\mathcal{S}_{u,\ell}^C = \{\mathbf{c}_{u,\ell}'^i\}_{i=0}^{r_\ell}$ be the set with the embeddings of one positive ($i = 0$) and r_ℓ negative ($1 \leq i \leq r_\ell$) samples (*i.e.*, centroids) for node u chosen among k_ℓ centroids. Using an inner product critic, CGC defines a hierarchical community-based contrastive loss $\mathcal{L}_C$ to be:

$$\mathcal{L}_C = \sum_{u=1}^{n} -\left(\frac{1}{L} \sum_{\ell=1}^{L} \log \frac{\exp(\mathbf{h}_u \cdot \mathbf{c}_{u,\ell}'^0/\tau)}{\sum_{v=0}^{r_\ell} \exp(\mathbf{h}_u \cdot \mathbf{c}_{u,\ell}'^v/\tau)} \right). \tag{5}$$

Multi-Level Contrastive Learning Objective. The above loss terms capture signals on the community structure at multiple levels, *i.e.*, individual node features ($\mathcal{L}_F$), neighboring nodes ($\mathcal{L}_H$), and hierarchically structured communities ($\mathcal{L}_C$). CGC jointly optimizes

$$\mathcal{L} = \lambda_F \mathcal{L}_F + \lambda_H \mathcal{L}_H + \lambda_C \mathcal{L}_C \tag{6}$$

where λ_F, λ_H, and λ_C are weights for the loss terms. Via multi-level noise 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 $\mathcal{E}$, we use a GNN with a mean aggregator,

$$\mathbf{h}_v^l = \text{ReLU}(\mathbf{W}_G \cdot \text{MEAN}(\{\mathbf{h}_v^{l-1}\} \cup \{\mathbf{h}_u^{l-1} \mid \forall u \in \mathcal{N}(v)\})) \tag{7}$$

where node v's embedding $\mathbf{h}_v^l$ from the l-th layer of $\mathcal{E}$ is obtained by averaging the embeddings of node v and its neighbors from the $(l-1)$-th layer, followed by a linear transformation $\mathbf{W}_G$ and ReLU non-linearity; $\mathbf{h}_v^0$ is initialized to be the input node features $\mathbf{f}_v$.

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 $\mathbf{H}$ produced by $\mathcal{E}$ (line 2), a clustering algorithm Π (*e.g.*, k-means) refines cluster centroids $\{\mathbf{C}_\ell\}$ and memberships $\{\Phi_\ell\}$ (lines 3-5). (2) Based on the updated cluster centroids and memberships, CGC computes the loss and optimizes model parameters (lines 6-7). In $\{k_\ell\}$, 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_{τ_i} arrives in a temporal graph stream $\mathcal{G} = \{G_{\tau_1}, \ldots, G_{\tau_{i-1}}\}$, node embeddings $\mathbf{H}_{i-1}$ and cluster memberships Φ_{i-1} that CGC learned from the snapshots until $(i-1)$-th time span are incrementally updated to reflect the new information in G_{τ_i}. 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 $G_{i:j}$ to denote a temporal graph that merges the snapshots $\{G_{\tau_i}, \ldots, G_{\tau_j}\}$, *i.e.*, $G_{i:j} = (V, E_{i:j})$ where $E_{i:j} = \bigcup_{o=i}^{j} E_{\tau_o}$. 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 $\mathbf{h}_{u,j-1}$ 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 features $\widetilde{\mathbf{F}}$, and take node u's embedding from the corrupted node embeddings $\mathcal{E}(G_{i:j-1}, \widetilde{\mathbf{F}})$ as the negative sample; multiple negative samples can be obtained by using multiple sets of corrupted node features. Let $\mathcal{S}_{u,j}^{T} = \{\mathbf{h}_{u,j-1}^{\prime i}\}_{i=0}^{r}$ be the set with the embeddings of one positive ($i = 0$) and r negative ($1 \le i \le r$) samples of node u for the j-th time span, again $\prime$ denoting the involvement of sampling. CGC defines a time-based contrastive loss $\mathcal{L}_T$ for time span j to be:

$$\mathcal{L}_T = \sum_{u=1}^{n} -\log \frac{\exp(\mathbf{h}_{u,j} \cdot \mathbf{h}_{u,j-1}^{\prime 0} / \tau)}{\sum_{v=0}^{r} \exp(\mathbf{h}_{u,j} \cdot \mathbf{h}_{u,j-1}^{\prime v} / \tau)} \tag{8}$$

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

$$\mathcal{L} = \lambda_F \mathcal{L}_F + \lambda_H \mathcal{L}_H + \lambda_C \mathcal{L}_C + \lambda_T \mathcal{L}_T. \tag{9}$$

4.2.2 Encoder Architecture. We extend the GNN encoder such that when it aggregates the neighborhood of a node, more weight is given to the neighbors that interacted with the node more recently. To this end, we adjust the weight of a neighbor based on the elapsed time since its latest interaction. Let $t_{(u,v)}$ denote the timestamp of an edge between nodes u and v, and let $t_v^{\max} = \max_{u \in \mathcal{N}(v)} \{t_{(u,v)}\}$, *i.e.*, the most recent timestamp when node v interacted with its neighbors. With ψ denoting a time decay factor between 0 and 1, we apply time decay to the embedding $\mathbf{h}_u$ of neighbor u as follows:

$$\mathrm{td}(\mathbf{h}_u) = \psi^{t_v^{\max} - t_{(u,v)}} \, \mathbf{h}_u. \tag{10}$$

Then for time-aware neighborhood aggregation, $\mathbf{h}_u$ in Equation (7) is replaced with its time decayed version $\mathrm{td}(\mathbf{h}_u)$.

4.2.3 Graph Stream Segmentation. Given a new graph snapshot, 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.

Algorithm 2 CGC Framework

```
Input: graph stream $\mathcal{G}$, input node feature matrix $\mathbf{F} \in \mathbb{R}^{n \times d'}$
Output: $\{$ cluster memberships $\mathbf{\Phi}_i \in \mathbb{R}^{n \times k}$, node embeddings
    $\mathbf{H}_i \in \mathbb{R}^{n \times d'}$, graph stream segment $\mathcal{G}_i^{\mathrm{seg}}\}$ for each time span $i$
1 $\mathcal{G}_0^{\mathrm{seg}} = \{\}$
2 for each $G_{\tau_i} \in \mathcal{G}$ do
3     $\mathcal{G}_i^{\mathrm{seg}}$ = GraphStreamSegmentation($G_{\tau_i}, \mathcal{G}_{i-1}^{\mathrm{seg}}, \mathbf{F}$)    ▹Alg. 3
4     $\mathbf{\Phi}_i, \mathbf{H}_i, \mathbf{C}_i$ = ContrastiveGraphClustering(Merge($\mathcal{G}_i^{\mathrm{seg}}$), $\mathbf{F}$)
      ▹Alg. 1
5 return $\{\mathbf{\Phi}_i, \mathbf{H}_i, \mathcal{G}_i^{\mathrm{seg}}\}_i$
```

Let $\mathcal{G}_{\mathrm{seg}} = \{G_{\tau_i}, \ldots, G_{\tau_j}\}$ be the current graph stream segment for some i and j ($i < j$). Given a new snapshot $G_{\tau_{j+1}}$, we expand the current segment $\mathcal{G}_{\mathrm{seg}}$ with $G_{\tau_{j+1}}$ if $G_{\tau_{j+1}}$ is similar to $\mathcal{G}_{\mathrm{seg}}$; if not, we start a new graph stream segment consisting only of $G_{\tau_{j+1}}$. 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 $\mathcal{G}_{\mathrm{seg}}$ and $G_{\tau_{j+1}}$. 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 $G_{\tau_{j+1}}$ will be similar to their embeddings in the existing segment $\mathcal{G}_{\mathrm{seg}}$ if $G_{\tau_{j+1}}$ is similar to $\mathcal{G}_{\mathrm{seg}}$. By the same token, a major change in the new snapshot will lead to a large difference between the embeddings of a node in $G_{\tau_{j+1}}$ and $\mathcal{G}_{\mathrm{seg}}$. Let V^* be the nodes appearing in both $\mathcal{G}_{\mathrm{seg}}$ and $G_{\tau_{j+1}}$. Let $\mathbf{H}_{V^*}^{\mathrm{seg}}$, $\mathbf{H}_{V^*}^{j+1} \in \mathbb{R}^{|V^*| \times d'}$ be the two sets of embeddings of the nodes in V^*, computed for $\mathcal{G}_{\mathrm{seg}}$ and $G_{\tau_{j+1}}$, respectively, as discussed above. Using a distance metric $d(\cdot,\cdot)$ (*e.g.*, cosine distance), we define the distance $\mathrm{Dist}(\cdot,\cdot)$ between $\mathbf{H}_{V^*}^{\mathrm{seg}}$ and $\mathbf{H}_{V^*}^{j+1}$ to be

$$\mathrm{Dist}(\mathbf{H}_{V^*}^{\mathrm{seg}}, \mathbf{H}_{V^*}^{t+1}) = \mathrm{MEAN}\{d((\mathbf{H}_{V^*}^{\mathrm{seg}})_i, (\mathbf{H}_{V^*}^{t+1})_i) \mid i \in V^*\} \tag{11}$$

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 language, 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 coauthored a paper. Node features are keywords of the author's papers, and node labels denote most active field of study of each author.

5.1.2 Temporal Datasets. Table 6 presents the statistics of temporal datasets. These datasets do not contain input node features, and dynamic node labels are available only for DBLP-T.

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 ARGVA [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. CTDNE [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, CTDNE), 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 difficult 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.

Table 3: CGC achieves the best node clustering results on static graphs. Best results are in bold, and second best results are underlined.

Method	DBLP-S				ACM				Citeseer				MAG-CS			
	ACC	NMI	ARI	F1	ACC	NMI	ARI	F1	ACC	NMI	ARI	F1	ACC	NMI	ARI	F1
K-means [22]	38.7±0.7	11.5±0.4	7.0±0.4	31.9±0.3	67.3±0.7	32.4±0.5	30.6±0.7	67.6±0.7	39.3±3.2	16.9±3.2	13.4±3.0	36.1±3.5	34.2±2.2	33.0±1.5	4.5±1.3	19.4±0.4
AE [23]	51.4±0.4	25.4±0.2	12.2±0.4	52.5±0.4	81.8±0.1	49.3±0.2	54.6±0.2	82.0±0.1	57.1±0.1	27.6±0.1	29.3±0.1	53.8±0.1	32.5±1.9	35.9±2.3	12.9±1.5	14.0±1.1
DEC [63]	58.2±0.6	29.5±0.3	23.9±0.4	59.4±0.5	84.3±0.8	54.5±1.5	60.6±1.9	84.5±0.7	55.9±0.2	28.3±0.3	28.1±0.4	52.6±0.2	44.4±3.4	53.5±2.8	33.6±4.0	28.4±3.1
IDEC [20]	60.3±0.6	31.2±0.5	25.4±0.6	61.3±0.6	85.1±0.5	56.6±1.2	62.2±1.5	85.1±0.5	60.5±1.4	27.2±2.4	25.7±2.7	61.6±1.4	45.7±1.8	55.3±2.6	33.5±3.4	30.8±2.3
SVD [15]	29.3±0.4	0.1±0.0	0.0±0.1	13.3±2.2	39.9±5.8	3.8±4.3	3.1±4.2	30.1±8.2	24.1±1.2	5.7±1.5	0.1±0.3	11.4±1.7	25.7±4.4	13.6±7.3	1.3±2.2	9.7±4.6
DGI [60]	32.5±2.4	3.7±1.8	1.7±0.9	29.3±3.3	88.0±1.1	63.0±1.9	67.7±2.5	88.0±1.0	64.1±1.3	38.8±1.2	38.1±1.9	60.4±0.9	60.0±0.6	65.9±0.4	50.3±0.9	47.3±0.4
GAE [28]	61.2±1.2	30.8±0.9	22.0±1.4	61.4±2.2	84.5±1.4	55.4±1.9	59.5±3.1	84.7±1.3	61.4±0.8	34.6±0.7	33.6±1.2	57.4±0.8	63.2±2.6	69.9±0.6	52.8±1.5	58.1±4.1
VGAE [28]	58.6±0.1	26.9±0.1	17.9±0.1	58.7±0.1	84.1±0.2	53.2±0.5	57.7±0.7	84.2±0.2	61.0±0.4	32.7±0.3	33.1±0.5	57.7±0.5	60.4±2.9	65.3±1.4	50.0±2.1	53.8±4.0
ARGA [40]	61.6±1.0	26.8±1.0	22.7±0.3	61.8±0.9	86.1±1.2	55.7±1.4	62.9±2.1	86.1±1.2	56.9±0.7	34.5±0.8	33.4±1.5	54.8±0.8	47.9±6.0	48.7±3.0	23.6±9.0	40.3±5.0
DAEGC [61]	62.1±0.5	32.5±0.5	21.0±0.5	61.8±0.7	86.9±2.8	56.2±4.2	59.4±3.9	87.1±2.8	64.5±1.4	36.4±0.9	37.8±1.2	62.2±1.3	48.1±3.8	60.3±0.8	47.4±4.2	32.2±3.2
SDCN [6]	68.1±1.8	39.5±1.3	39.2±2.0	67.7±1.5	90.5±0.2	68.3±0.3	73.9±0.4	90.4±0.2	66.0±0.3	38.7±0.3	40.2±0.4	63.6±0.2	51.6±5.5	58.0±1.9	46.9±8.1	30.2±4.3
AGCN [48]	73.3±0.4	39.7±0.4	42.5±0.3	72.8±0.6	90.6±0.2	68.4±0.5	74.2±0.4	90.6±0.2	68.8±0.2	41.5±0.3	43.8±0.3	62.4±0.2	54.2±5.2	59.4±2.1	49.2±6.5	36.3±4.4
CGC (Ours)	**77.6±0.5**	**46.1±0.6**	**49.7±1.1**	**77.2±0.4**	**92.3±0.3**	**72.9±0.7**	**78.4±0.6**	**92.3±0.3**	**69.6±0.6**	**44.6±0.6**	**46.0±0.6**	**65.5±0.7**	**69.3±4.0**	**79.3±1.2**	**64.4±3.7**	**62.1±4.5**

Table 4: CGC achieves the highest node clustering accuracy on the temporal DBLP-T graph. Best results are in bold, and second best results are underlined.

Method	DBLP-T			
	ACC	NMI	ARI	F1
SVD [15]	61.60±0.01	0.16±0.02	-0.06±0.01	38.13±0.02
SVD-latest	61.62±0.02	0.16±0.02	-0.04±0.02	38.17±0.04
DGI [60]	61.64±0.02	0.06±0.01	0.08±0.01	38.77±0.07
DGI-latest	61.66±0.02	0.06±0.02	0.03±0.02	38.44±0.06
GAE [28]	63.76±0.18	4.40±0.16	7.28±0.25	59.75±0.20
GAE-latest	60.17±0.04	0.72±0.02	2.47±0.05	52.36±0.11
VGAE [28]	60.06±0.18	1.63±0.06	3.44±0.11	55.66±0.11
VGAE-latest	60.67±0.03	0.77±0.02	2.61±0.03	51.90±0.06
ARGA [40]	58.46±0.25	0.16±0.04	0.86±0.16	48.95±0.27
ARGA-latest	60.54±0.13	0.19±0.05	0.81±0.15	45.37±0.30
SDCN [6]	56.70±0.60	2.18±0.72	2.88±0.51	55.66±0.87
SDCN-latest	51.51±0.26	0.13±0.03	0.11±0.04	50.79±0.30
AGCN [48]	56.04±0.86	0.88±0.38	1.11±0.40	50.34±1.13
AGCN-latest	54.52±0.91	0.09±0.03	0.14±0.12	48.67±0.85
CTDNE [38]	51.58±0.07	1.98±0.06	-0.99±0.03	48.19±0.27
CTDNE-latest	50.57±0.10	0.02±0.01	0.01±0.01	49.85±0.10
TIMERS [68]	61.70±0.00	0.09±0.01	0.02±0.00	38.21±0.01
DynGEM [17]	60.73±0.12	0.27±0.04	1.26±0.12	46.52±0.22
DynAERNN [16]	62.34±0.09	0.69±0.08	1.66±0.13	44.83±0.22
EvolveGCN [41]	61.02±0.00	0.79±0.00	2.64±0.00	51.16±0.02
CTGCN [35]	59.07±0.47	1.06±0.12	2.88±0.27	55.14±0.23
CGC (Ours)	**71.82±0.99**	**21.87±1.85**	**27.28±2.93**	**71.12±0.86**

5.4 Temporal Link Prediction Accuracy (RQ2)

The task is to predict the graph $G_{\tau'} = (V, E_{\tau'})$ for the next time span τ', where $E_{\tau'}$ are the temporal positive (*i.e.*, observed) edges. We uniformly randomly sample the same amount of temporal negative edges $E^-_{\tau'}$ such that $E^-_{\tau'} = \{(u,v) \mid u, v \sim \text{Uniform}(1, \ldots, n) \wedge (u,v) \notin E_{\tau'}\}$. Given an edge $(u,v) \in E_{\tau'} \cup E^-_{\tau'}$ for time span τ' to predict, we estimate the likelihood of such an edge existing as $A^{\tau'}_{uv} = \boldsymbol{\phi}_u^\intercal \boldsymbol{\phi}_v$, where $\boldsymbol{\phi}_u$ and $\boldsymbol{\phi}_v$ are cluster memberships for nodes u and v. We can use link prediction task for evaluating clustering quality, since nodes in the same cluster are more likely to form a link between them than nodes belonging to different clusters. Also, since temporal link prediction is based on the time-evolving membership vector $\boldsymbol{\phi}$, 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.*, $E_{\tau'} \cup E^-_{\tau'}$) changes over time, we average the performance for each snapshot, weighted by the size of $E_{\tau'} \cup E^-_{\tau'}$. 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 statc 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 contrastive 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 (λ_F), network homophily (λ_H), and hierarchical communities (λ_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+H on ACM and H to H+C on DBLP-S. Especially, the best result on ACM and DBLP-S are obtained when all objectives are used together (F+H+C). However, DBLP-S shows a different pattern, where the best result was obtained with F+C. Notably, in DBLP-S, the objective on network homophily was not useful whether it is used alone (H) or with others (F vs. F+H). In DBLP-S, 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 graph clustering and community detection, 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
VGAE [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
VGAE-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
ARGVA [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	26.01±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	44.24±2.2	54.23±2.2	54.43±1.5
AGCN-latest	41.24±3.2	49.01±2.5	41.44±5.8	51.27±4.0	51.81±1.1	52.87±0.4
CTDNE [38]	7.06±0.0	31.55±0.0	16.97±0.3	33.59±0.1	54.73±0.1	54.16±0.1
CTDNE-latest	7.27±0.0	32.28±0.0	7.36±0.1	31.98±0.0	50.11±0.0	52.70±0.1
TIMERS [68]	23.84±0.2	37.02±0.1	15.09±0.1	33.72±0.0	48.87±0.1	49.65±0.1
DynGEM [17]	26.65±0.8	36.61±0.3	25.52±2.8	36.24±0.9	47.46±0.5	46.69±0.4
DynAERNN [16]	26.17±2.1	41.39±1.6	18.23±1.1	40.15±0.7	44.81±2.0	50.44±2.1
EvolveGCN [41]	23.79±1.0	47.45±0.1	24.67±0.6	46.45±0.2	47.00±0.9	47.08±0.4
CTGCN [35]	50.58±2.4	54.54±1.5	51.61±4.5	57.56±2.8	75.51±0.9	76.82±0.7
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**

modularity-based methods [14], METIS [26], spectral methods [3], methods based on SVD [15], connected components [42, 43], tensor factorization [19, 39, 45, 46] and MDL (Minimum Description Length) [1, 55]. However, these methods all miss one or more of the desiderata of Table 1, as they mostly focus on utilizing the graph structure alone, with no support for input node features or the time evolution of graphs, and without learning node representations, which are useful for downstream applications. Our comparison with SVD [15], one of the representative methods for community detection, shows the benefits of satisfying the desiderata in Table 1.

In this paper, we focus on another group of methods for graph clustering, namely, deep graph clustering (DGC). Methods for DGC can be grouped into two categories: (1) two-stage methods that perform clustering after learning representations, and (2) single-stage methods that jointly perform clustering and representation learning (RL). Unsupervised graph RL methods are used for two-stage deep graph clustering (DGC). In [57], for instance, AEs are used to learn non-linear node embeddings, and then K-means is applied to get clustering assignments. GNN-based encoders are adopted in more recent methods. GAE [28] and VGAE [28] learn node embeddings using a graph autoencoder and a variational variant. ARGA [40] and ARGVA [40] employ an adversarially regularized graph autoencoder and its variational version. A few recent studies [56, 60, 62, 66] investigated self-supervised learning techniques for graph RL, *e.g.*, DGI [60] optimizes GCN encoder by contrasting node embeddings with the embedding of the graph.

DMoN [58] is a single-stage method that performs clustering via spectral modularity maximization. DAEGC [61] simultaneously optimizes embedding learning and graph clustering by combining the clustering loss of DEC with the graph reconstruction loss of graph attentional AEs. SDCN [6] improves DAEGC by integrating a GCN encoder and AEs via a delivery operator. AGCN [48] further improves upon SDCN by developing two attention-based fusion modules, which aggregate features from GCNs and AEs, and multi-scale features from different layers. Despite some differences (*e.g.*, encoder architectures), existing DGC methods are mainly based on AEs, involve reconstruction loss minimization, and use the same clustering objective [63] with small adjustments. The proposed CGC performs deep graph clustering in a novel contrastive graph learning framework with multi-level contrastive objectives.

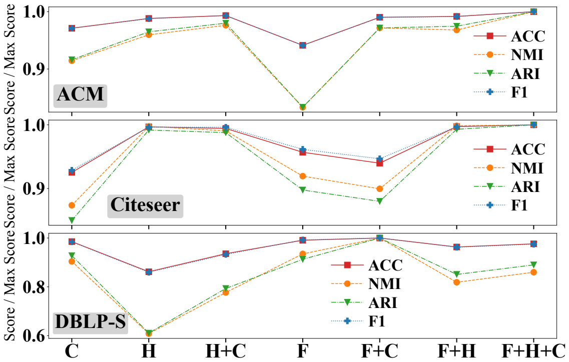


Figure 2: Node clustering performance of CGC, obtained with different contrastive objectives. F: Node Features. H: Network Homophily. C: Hierarchical Communities.

Temporal Graph Clustering (TGC). Existing methods mainly perform TGC based on the graph structure and its temporal change, without considering node features and their semantics in the clustering objective. Existing TGC methods can be grouped into two classes: snapshot clustering [5, 8, 18] and consensus clustering [2, 10, 31, 51, 52]. Given graph snapshots, each snapshot is clustered separately in snapshot clustering, thereby ignoring inter-snapshot information. Consensus clustering instead finds a single partitioning for the entire graph snapshots. Consensus and snapshot clustering correspond to two fixed choices (*i.e.*, the entire snapshots vs. the last one), which is not always optimal. CGC instead adaptively determines a subset of snapshots to find clusters from.

For two-stage deep TGC, unsupervised dynamic graph representation learning methods can also be employed, which learn dynamic embeddings using temporal random walk [38], incremental SVD [68], AEs [16, 17], and RNNs combined with GCNs [35, 41, 44]. Yet no single-stage DGC methods have been designed for TGC. This paper presents the first such method for temporal network analysis.

7 CONCLUSION

This work presented CGC, a new deep graph clustering framework for community detection and tracking in the web data.

- **Novel Framework.** CGC jointly learns node embeddings and cluster memberships in a novel contrastive graph learning framework. CGC effectively finds clusters by using information along multiple dimensions, *e.g.*, node features, hierarchical communities.
- **Temporal Graph Clustering.** CGC is designed to find clusters from time-evolving graphs, improving upon existing deep graph clustering methods, which are designed for static graphs.
- **Effectiveness.** We show the effectiveness of CGC via extensive evaluation on several static and temporal real-world graphs.

REFERENCES

[1] Leman Akoglu, Hanghang Tong, Brendan Meeder, and Christos Faloutsos. 2012. PICS: Parameter-free Identification of Cohesive Subgroups in Large Attributed Graphs. In *SDM*. SIAM / Omnipress, 439–450.

[2] Thomas Aynaud and Jean-Loup Guillaume. 2011. Multi-step community detection and hierarchical time segmentation in evolving networks. In *Proceedings of the 5th SNA-KDD workshop*, Vol. 11.

[3] Stephen T. Barnard and Horst D. Simon. 1993. A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems. In *PPSC*. SIAM, 711–718.

[4] Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeswar, Sherjil Ozair, Yoshua Bengio, R. Devon Hjelm, and Aaron C. Courville. 2018. Mutual Information Neural Estimation. In *ICML (Proceedings of Machine Learning Research, Vol. 80)*. PMLR, 530–539.

[5] Tanya Y. Berger-Wolf and Jared Saia. 2006. A framework for analysis of dynamic social networks. In *KDD*. ACM, 523–528.

[6] Deyu Bo, Xiao Wang, Chuan Shi, Meiqi Zhu, Emiao Lu, and Peng Cui. 2020. Structural Deep Clustering Network. In *WWW*. ACM / IW3C2, 1400–1410.

[7] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. 2020. A Simple Framework for Contrastive Learning of Visual Representations. In *ICML (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 1597–1607.

[8] Yun Chi, Xiaodan Song, Dengyong Zhou, Koji Hino, and Belle L. Tseng. 2007. Evolutionary spectral clustering by incorporating temporal smoothness. In *KDD*. ACM, 153–162.

[9] CiteSeer. 2021 [Online]. https://citeseerx.ist.psu.edu. Accessed: 2021-10-01.

[10] Joseph Crawford and Tijana Milenković. 2018. ClueNet: Clustering a temporal network based on topological similarity rather than denseness. *PLOS ONE* 13, 5 (05 2018), 1–25.

[11] DBLP. 2021 [Online]. https://dblp.org. Accessed: 2021-10-01.

[12] Foursquare. 2021 [Online]. https://foursquare.com. Accessed: 2021-10-01.

[13] PyTorch Geometric. 2021. PyG. https://github.com/pyg-team/pytorch_geometric. Accessed: 2021-10-20.

[14] Michelle Girvan and Mark EJ Newman. 2002. Community structure in social and biological networks. *Proceedings of the national academy of sciences* 99, 12 (2002), 7821–7826.

[15] Gene H Golub and Christian Reinsch. 1971. Singular value decomposition and least squares solutions. In *Linear algebra*. Springer, 134–151.

[16] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. 2020. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowl. Based Syst.* 187 (2020).

[17] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. 2018. DynGEM: Deep Embedding Method for Dynamic Graphs. *CoRR* abs/1805.11273 (2018).

[18] Derek Greene, Dónal Doyle, and Padraig Cunningham. 2010. Tracking the Evolution of Communities in Dynamic Social Networks. In *ASONAM*. IEEE Computer Society, 176–183.

[19] Ekta Gujral, Ravdeep Pasricha, and Evangelos E. Papalexakis. 2020. Beyond Rank-1: Discovering Rich Community Structure in Multi-Aspect Graphs. In *WWW*. ACM / IW3C2, 452–462.

[20] Xifeng Guo, Long Gao, Xinwang Liu, and Jianping Yin. 2017. Improved Deep Embedded Clustering with Local Structure Preservation. In *IJCAI*. ijcai.org, 1753–1759.

[21] Michael Gutmann and Aapo Hyvärinen. 2010. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *AISTATS (JMLR Proceedings, Vol. 9)*. JMLR.org, 297–304.

[22] John A Hartigan and Manchek A Wong. 1979. Algorithm AS 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)* 28, 1 (1979), 100–108.

[23] Geoffrey E Hinton and Ruslan R Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *science* 313, 5786 (2006), 504–507.

[24] R. Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Philip Bachman, Adam Trischler, and Yoshua Bengio. 2019. Learning deep representations by mutual information estimation and maximization. In *ICLR*. OpenReview.net.

[25] Zhuxi Jiang, Yin Zheng, Huachun Tan, Bangsheng Tang, and Hanning Zhou. 2017. Variational Deep Embedding: An Unsupervised and Generative Approach to Clustering. In *IJCAI*. ijcai.org, 1965–1972.

[26] George Karypis and Vipin Kumar. 1998. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.* 20, 1 (1998), 359–392.

[27] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. 2020. Supervised Contrastive Learning. *CoRR* abs/2004.11362 (2020).

[28] Thomas N. Kipf and Max Welling. 2016. Variational Graph Auto-Encoders. *CoRR* abs/1611.07308 (2016).

[29] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR (Poster)*. OpenReview.net.

[30] Harold W Kuhn. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly* 2, 1-2 (1955), 83–97.

[31] Andrea Lancichinetti and Santo Fortunato. 2012. Consensus clustering in complex networks. *Scientific reports* 2, 1 (2012), 1–7.

[32] Peizhao Li, Han Zhao, and Hongfu Liu. 2020. Deep Fair Clustering for Visual Learning. In *CVPR*. Computer Vision Foundation / IEEE, 9067–9076.

[33] ACM Digital Library. 2021 [Online]. https://dl.acm.org. Accessed: 2021-10-01.

[34] Deep Graph Library. 2021. DGI. https://github.com/dmlc/dgl/tree/master/examples/pytorch/dgi. Accessed: 2021-10-20.

[35] J. Liu, C. Xu, C. Yin, W. Wu, and Y. Song. 2020. K-Core based Temporal Graph Convolutional Network for Dynamic Graphs. *IEEE Transactions on Knowledge and Data Engineering* (2020), 1–1. https://doi.org/10.1109/TKDE.2020.3033829

[36] Rui Lu, Zhiyao Duan, and Changshui Zhang. 2019. Audio-Visual Deep Clustering for Speech Separation. *IEEE ACM Trans. Audio Speech Lang. Process.* 27, 11 (2019), 1697–1712.

[37] Naveen Sai Madiraju, Seid M. Sadat, Dimitry Fisher, and Homa Karimabadi. 2018. Deep Temporal Clustering : Fully Unsupervised Learning of Time-Domain Features. *CoRR* abs/1802.01059 (2018).

[38] Giang Hoang Nguyen, John Boaz Lee, Ryan A. Rossi, Nesreen K. Ahmed, Eunyee Koh, and Sungchul Kim. 2018. Continuous-Time Dynamic Network Embeddings. In *WWW (Companion Volume)*. ACM, 969–976.

[39] Sejoon Oh, Namyong Park, Lee Sael, and U Kang. 2018. Scalable Tucker Factorization for Sparse Tensors - Algorithms and Discoveries. In *ICDE*. IEEE Computer Society, 1120–1131.

[40] Shirui Pan, Ruiqi Hu, Sai-Fu Fung, Guodong Long, Jing Jiang, and Chengqi Zhang. 2020. Learning Graph Embedding With Adversarial Training Methods. *IEEE Trans. Cybern.* 50, 6 (2020), 2475–2487.

[41] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B. Schardl, and Charles E. Leiserson. 2020. EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs. In *AAAI*. AAAI Press, 5363–5370.

[42] Ha-Myung Park, Namyong Park, Sung-Hyon Myaeng, and U Kang. 2016. Partition Aware Connected Component Computation in Distributed Systems. In *ICDM*. IEEE Computer Society, 420–429.

[43] Ha-Myung Park, Namyong Park, Sung-Hyon Myaeng, and U Kang. 2020. PACC: Large scale connected component computation on Hadoop and Spark. *PLOS ONE* 15, 3 (03 2020), 1–25. https://doi.org/10.1371/journal.pone.0229936

[44] Namyong Park, Fuchen Liu, Purvanshi Mehta, Dana Cristofor, Christos Faloutsos, and Yuxiao Dong. 2022. EvoKG: Jointly Modeling Event Time and Network Structure for Reasoning over Temporal Knowledge Graphs. In *WSDM*. ACM.

[45] Namyong Park, Sejoon Oh, and U Kang. 2017. Fast and Scalable Distributed Boolean Tensor Factorization. In *ICDE*. IEEE Computer Society, 1071–1082.

[46] Namyong Park, Sejoon Oh, and U Kang. 2019. Fast and scalable method for distributed Boolean tensor factorization. *VLDB J.* 28, 4 (2019), 549–574.

[47] Xi Peng, Shijie Xiao, Jiashi Feng, Wei-Yun Yau, and Zhang Yi. 2016. Deep Subspace Clustering with Sparsity Prior. In *IJCAI*. IJCAI/AAAI Press, 1925–1931.

[48] Zhihao Peng, Hui Liu, Yuheng Jia, and Junhui Hou. 2021. Attention-driven Graph Clustering Network. In *ACM Multimedia*. ACM, 935–943.

[49] Ben Poole, Sherjil Ozair, Aäron van den Oord, Alex Alemi, and George Tucker. 2019. On Variational Bounds of Mutual Information. In *ICML (Proceedings of Machine Learning Research, Vol. 97)*. PMLR, 5171–5180.

[50] Yahoo Webscope Program. 2021 [Online]. https://webscope.sandbox.yahoo.com. Accessed: 2021-10-01.

[51] Martin Rosvall and Carl T Bergstrom. 2008. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences* 105, 4 (2008), 1118–1123.

[52] Martin Rosvall and Carl T Bergstrom. 2011. Multilevel compression of random walks on networks reveals hierarchical organization in large integrated systems. *PloS one* 6, 4 (2011), e18209.

[53] scikit learn. 2021. scikit-learn. https://github.com/scikit-learn/scikit-learn. Accessed: 2021-10-20.

[54] Uriel Singer. 2021. CTDNE. https://github.com/urielsinger/CTDNE.

[55] Jimeng Sun, Christos Faloutsos, Spiros Papadimitriou, and Philip S. Yu. 2007. GraphScope: parameter-free mining of large time-evolving graphs. In *KDD*. ACM, 687–696.

[56] Ke Sun, Zhouchen Lin, and Zhanxing Zhu. 2020. Multi-Stage Self-Supervised Learning for Graph Convolutional Networks on Graphs with Few Labeled Nodes. In *AAAI*. AAAI Press, 5892–5899.

[57] Fei Tian, Bin Gao, Qing Cui, Enhong Chen, and Tie-Yan Liu. 2014. Learning Deep Representations for Graph Clustering. In *AAAI*. AAAI Press, 1293–1299.

[58] Anton Tsitsulin, John Palowitch, Bryan Perozzi, and Emmanuel Müller. 2020. Graph clustering with graph neural networks. *arXiv preprint arXiv:2006.16904* (2020).

[59] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation Learning with Contrastive Predictive Coding. *CoRR* abs/1807.03748 (2018).

[60] Petar Velickovic, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. 2019. Deep Graph Infomax. In *ICLR (Poster)*. OpenReview.net.

[61] Chun Wang, Shirui Pan, Ruiqi 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] Junyuan Xie, 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 Shen. 2020. When Does Self-Supervision Help Graph Convolutional Networks?. In *ICML (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 10871–10880.

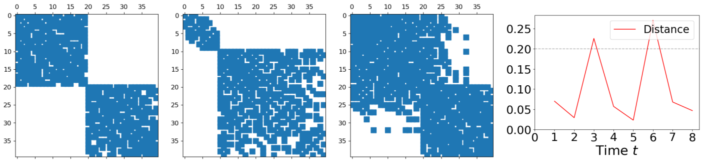
[67] Mingxuan Yue, Yaguang Li, Haoze Yang, Ritesh Ahuja, Yao-Yi Chiang, and Cyrus Shahabi. 2019. DETECT: Deep Trajectory Clustering for Mobility-Behavior Analysis. In *IEEE BigData*. IEEE, 988–997.

[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.



(a) Segment before the change point (time 3)

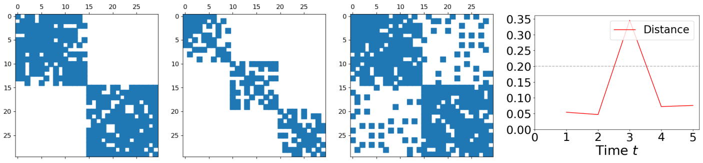
(b) Segment after segmentation at the change point

(c) Segment with no segmentation at change point

(d) Distance between G_t and the existing segment 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.



(a) Segment before the change point (time 3)

(b) Segment after segmentation at the change point

(c) Segment with no segmentation at change point

(d) Distance between G_t and the existing segment over time

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)).

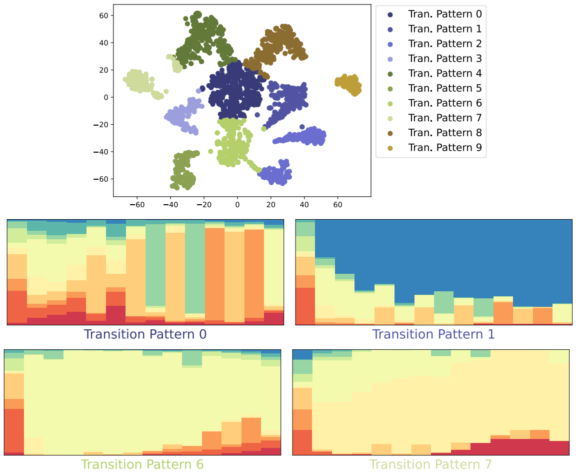


Figure 5: Node clusters (top) based on their transition patterns (bottom) in the Yahoo-Msg dataset.

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

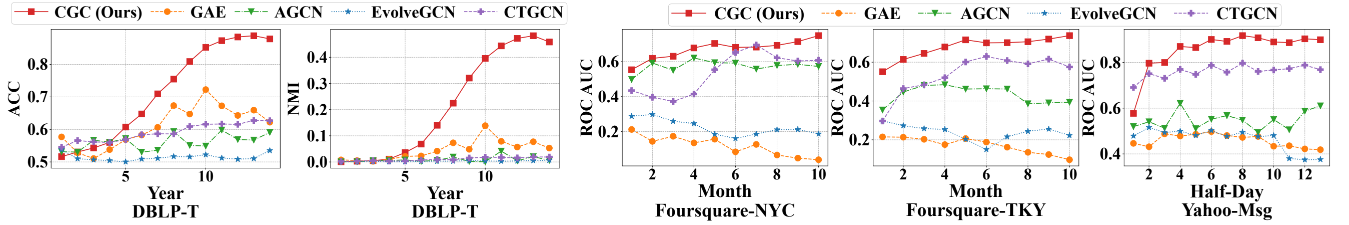
Figure 6 shows how the performance of CGC and four select baselines changes over time. For static baselines, we report the results obtained by clustering all observed graph snapshots at each time step.

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



(a) Temporal Node Clustering

(b) Temporal Link Prediction

Figure 6: CGC achieves the best clustering performance nearly consistently on temporal graphs over the entire time period.

Table 6: Summary of temporal real-world datasets. N/A denotes that the corresponding datasets do not have node labels.

Dataset	Edge Type (node i, node j, time t)	# Nodes	# Edges	Time Range (Inclusive)	# Graph Snapshots	Snapshot Interval	# Dynamic Node Classes
Yahoo-Msg	(user, user, time-second)	82,309 (82,309 users)	786,911	0-6 (days)	14	12 hours	N/A
Foursquare-NYC	(user, venue, time-second)	39,416 (1,083 users, 38,333 venues)	454,856	0-318 (days)	11	30 days	N/A
Foursquare-TKY	(user, venue, time-second)	64,151 (2,293 users, 61,858 venues)	1,147,406	0-318 (days)	11	30 days	N/A
DBLP-T	(author, author, time-year)	6,942 (6,942 authors)	168,124	0-13 (years)	14	1 year	2

Table 7: Summary of static real-world datasets used in experiments. In all datasets, nodes have labels and input features.

Dataset	Edge Type (node i, node j)	# Nodes	# Edges	# Node Classes	Feature Dimension
ACM	(paper, paper)	3,025	26,256	3	1,870
DBLP-S	(author, author)	4,057	7,056	4	334
Citeseer	(document, document)	3,327	9,104	6	3,703
MAG-CS	(author, author)	18,333	163,788	15	6,805

DGI [34]. We use a single-layer GCN in DGI as in the open source code [34]. For CGC, we set node embedding size to 200, and used Adam optimizer with a weight decay of 0.0001. We set the learning rate to 0.0005 (Citeseer), 0.001 (ACM, MAG-CS), and 0.005 (DBLP-S). We used a single layer GNN in CGC. We set temperature τ to 0.65, δ to 0.7, and update interval $R = 2$ in all experiments. Let r_F, r_H, and r_C^ℓ be the number of negatives per positive sample for the contrastive loss $\mathcal{L}_F$, $\mathcal{L}_H$, and $\mathcal{L}_C$, where ℓ in r_C^ℓ refers to the ℓ-th level clusters. We set r_F to 180 (MAG-CS), 50 (DBLP-S), and 30 (ACM, Citeseer); r_H to 60 (MAG-CS) and 10 (others); r_C^ℓ to 60 (MAG-CS) and 30 (others) for each ℓ. Let k denote the number of clusters to find. We set $\mathcal{K} = \{k, 5k, 25k\}$. For DBLP-S, we set $\lambda_F = 4$, $\lambda_H = 0$, $\lambda_C = 1$. For ACM, Citeseer, and MAG-CS, we set $\lambda_F = 1$, $\lambda_H = 1$, $\lambda_C = 1$.

Experiments for Temporal Data. Since the temporal graphs used in experiments have no input node features $\mathbf{F}$, we used learnable node embeddings as the input node features, which were initialized by applying SVD to the row normalized adjacency matrix.

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 SVD and DGI, we used the same setting used for static graphs. For GAE, VGAE, ARGA, and ARGVA, we used the implementation of the PyTorch Geometric [13] with two-layer GCN encoders. For SDCN 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 θ in TIMERS to 0.17. In DynGEM, we set α to 10^{-5}, β to 10, and both ν_1 and ν_2 to 10^{-4}. For DynAERNN, we set β to 5, the look back parameter to 3, and both ν_1 and ν_2 to 10^{-6}. In EvolveGCN, we used a two-layer GCRN; 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 $\lambda_H = 1$, $\lambda_C = \lambda_T = 0.2$, $\lambda_F = 0$; $\psi = 0.99$, $\theta = 0.3$. Let r_T be the number of negatives per positive sample for the loss $\mathcal{L}_T$. For all temporal datasets, we set $r_F = r_H = r_T = 10$. We set r_C^ℓ to 60 (link prediction datasets) and 30 (DBLP-T) for each ℓ. We set $\mathcal{K} = \{5k, 25k\}$ for DBLP-T, and $\mathcal{K} = \{k, 5k, 25k\}$ 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 $\mathcal{G}_{\text{seg}}$, new graph $G_{\tau_{j+1}}$ for time span
    $j+1$, input node features $\mathbf{F} \in \mathbb{R}^{n \times d}$, segmentation threshold $\theta$
Output: graph stream segment $\mathcal{G}_{\text{seg}}$
if $\mathcal{G}_{\text{seg}} \neq \varnothing$ then
    $\mathcal{G}_{\text{seg}} = \text{Merge}(\mathcal{G}_{\text{seg}})$
    $V^* = \text{Nodes}(\mathcal{G}_{\text{seg}}) \cap \text{Nodes}(G_{\tau_{j+1}})$
    $\mathbf{H}^{\text{seg}} = \mathcal{E}(\mathcal{G}_{\text{seg}}, \mathbf{F})$
    $\mathbf{H}^{j+1} = \mathcal{E}(G_{\tau_{j+1}}, \mathbf{F})$
if $\mathcal{G}_{\text{seg}} = \varnothing$ or $\text{Dist}(\mathbf{H}^{\text{seg}}_{V^*}, \mathbf{H}^{j+1}_{V^*}) > \theta$ then
    $\mathcal{G}_{\text{seg}} = \{G_{\tau_{j+1}}\}$            ▷Start a new graph stream segment.
else
    $\mathcal{G}_{\text{seg}} = \mathcal{G}_{\text{seg}} \cup \{G_{\tau_{j+1}}\}$    ▷Add $G_{\tau_{j+1}}$ to the current segment.
return $\mathcal{G}_{\text{seg}}$
```

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].