Fuzzy Systems and Data Mining X
A.J. Tallón-Ballesteros (Ed.)

doi:10.3233/FAIA241413

A Graph Neural Network-Based Community Search System over Dynamic Graph

Yixin Song, Lihua Zhou[1]
Information Science and Engineering, Yunnan University, Kunming 650091, China

Abstract. Community search (CS), aiming to find a densely connected subgraph containing given query vertices, is an important research topic in social network analysis and is widely used in similar recommendation, team organization, friendship recommendation and other practical applications. The purpose of the CS system is to display searched community in a visual form to users. It can help users better understand and analyze networks, making better decisions. However, the exist CS systems are mostly designed for static graphs, they cannot capture the dynamic attributes and cannot intuitively display the dynamic changes of the community. In this paper, we develop a CS system over dynamic graph based on graph neural network (GNN), aiming to locate the community with cohesive attributes over dynamic graph and visualize the community to intuitively display the dynamic changes of vertices and the relationships between them. We design a GNN-based method to capture the dynamic changes of attributes and design a friendly front-end interface that visualizes the result community in the form of a timeline. It allows users to view the status of the result community at any snapshot and fine-tune the result community according to their own conditions.

Keywords. Community search system, graph neural network, dynamic graphs, attribute cohesiveness

1. Introduction

Community search (CS)[1-3] is an important research topic in graph analysis, aiming to search for a structure cohesive subgraph with query vertices that satisfies the given conditions. However, the results of CS are relatively abstract, and it's difficult to intuitively compare the advantages and disadvantages of two result communities. CS system is a system that uses visualization technology to graphically display the community. It receives the query conditions specified by users and displays the result community to users visually. Thus, CS systems can help researchers and users to explore the graph data and the result community. For this reason, Chen and Gao[4] proposed a visual interactive system (VICS-GNN) for CS with similar attributes via GNN. VICS-GNN is the first system to support interactive, flexible CS guided by user's labeling. However, VICS-GNN cannot display the dynamic changes of the communities, because it is designed for static graphs.

In this paper, we develop a CS system over dynamic graph to search for a community with similar attributes and allows users to intuitively explore the dynamic changes of

[1] Corresponding Author: Lihua Zhou, Information Science and Engineering, Yunnan University, Kunming 650091, China; E-mail: lhzhou@ynu.edu.cn.

attributes and relations between vertices. To the best of our knowledge, our proposed system is the first CS system over dynamic graphs.

Our proposed system contains a front-end and a back-end. The front-end designs a friendly visualization interface for users to explore result community easily and intuitively. Generally, a dynamic graph is a sequence of static graphs (snapshots), which represents the dynamic changes of edges, vertices and the related attributes. To visualize the dynamic changes of vertices and edges, we design a timeline to show the community at each snapshot, which allows user to explore the result in any snapshots. The back-end aims to search community with similar attributes by capturing the dynamic change of vertices' attributes. The dynamic changes of vertices' attributes reflect more comprehensive characteristics of real entities in real-world applications. For example, a user prefers action movies for a while, and then prefers science fiction movies after a while. So how to capture the dynamic changes of vertices' attributes is of great significance for CS.

Unfortunately, the most of CS methods over dynamic graphs are only consider the structure, ignoring the attributes of vertices. Although attribute community search (ACS)[5-7] can consider the attributes of vertices for CS, they cannot handle the dynamic changes of attributes. In addition, most of ACS methods are searching communities based on specified keywords, which means they cannot handle the non-linear attributes. For example, the keywords "Natural Language Processing" and "Machine Learning" are totally different, although both are closely related to the "Artificial Intelligence".

Due to the effectiveness of GNN in handling the non-linear attributes, we design a GNN based method as the default method of the back-end to capture the dynamic changes of attributes. Of course, other CS methods over dynamic graphs can also be used in the back-end.

2. System Overview

In this section, we give the framework of the CS system we proposed and introduce each module of it. The framework of the proposed CS system is shown in Figure 1, which consists of front-end and back-end. The front-end is used for a user to configure the parameters and visualize the results of CS. The parameters configured by the front-end are provided to the back-end for preprocessing and conducting subsequent community searches. The back-end is used to mine the dynamic changes of attributes, and performing the community search.

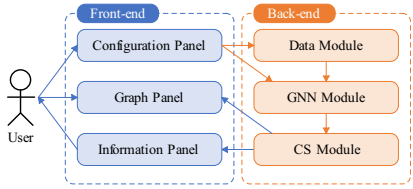


Figure 1. The framework of our CS system.

2.1. Front-end

We design a friendly visual interface for users to explore the dynamic community in a form of timeline. Figure 2(a) shows the front-end of the CS system we proposed, which includes 3 panels, namely configuration panel (box 1), graph panel (box 2) and information panel (box 3). Each panel is described as below.

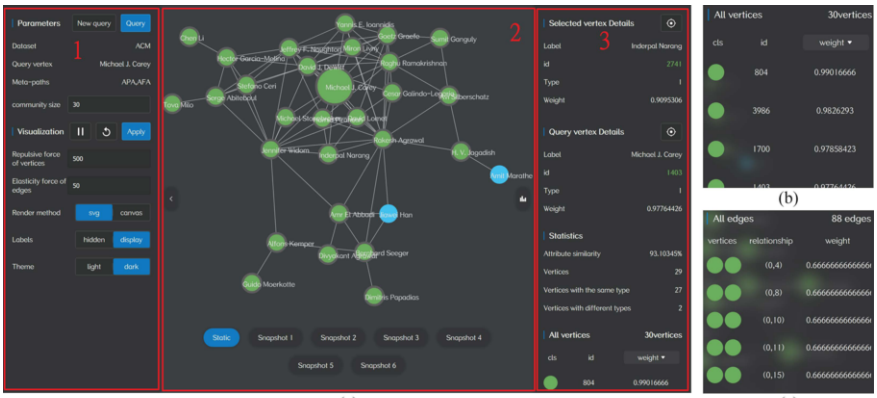


Figure 2. The front-end of our CS system (a) and the list of vertices and edges (b-c).

Configuration Panel. It allows users to set the parameters of CS and visualization. Once the user creates a new query, he/she needs to upload the dataset (a dynamic graph), set the query vertex and the meta-paths (if the uploaded dynamic graph is heterogeneous), and these three parameters will not change in this query. Then, the user sets the community size and the parameters of visualization to driving the back-end for CS. The community size constraint is commonly used in practical applications[8], such as only accepting a certain number of people to join a team due to the budget constraints.

Graph Panel. It provides users with a visual interface for exploring the result community in a form of timeline. It displays the result community in a static graph by default. The static graph contains the vertices and edges of all snapshots. Different colors in the graph represent different attribute categories of vertices. Floating below the graph panel is the timeline. It arranges all snapshots in chronological order. Users can choose to view the community status of any snapshot.

Information Panel. It shows the detail information of the vertex that the user selected in graph panel and the query vertex. It shows statistical information on the vertices and edges of the result community, which can help users judge the reliability of the results on a macroscopic level. In addition, the information of all vertices and edges in result community are also displayed to the user in the form of a list (Figure 2(b-c)).

2.2. Back-end

The back-end contains three modules and it does not involve the database, because the data is uploaded from users. Each module of the back-end is described as below.

Data Module. This module is responsible for processing the data uploaded by the user into a unified dynamic graph data structure. A dynamic graph data structure is a sequence of snapshots, and every snapshot is a static graph with vertices and edges in a specific time. If the snapshots in a dynamic graph are heterogeneous graphs, it will transform the heterogeneous graph into a homogeneous graph based on multiple symmetric meta-paths, i.e., only retain one type of vertices and edges, the edges represent the relationships between vertices with respect to multiple symmetric meta-paths.

GNN Module. This module aiming to capture the dynamic changes of attributes and calculates the attribute similarity between each vertex by GNN-based method. GNN[9] can learn high-dimensional embeddings of vertices by capturing attributes and structural information over a static graph. GNN has many variants, such as GCN[10], GAT[11] etc.

To capture the dynamic changes of attributes and structures over a dynamic graph, this module uses a two-level (spatial- and temporal-level) GAT model to fuse the embeddings of vertices from spatial and temporal perspectives respectively.

Specifically, let $\mathbf{H}^t \in \mathbb{R}^{N\times M}$ represents the initial embeddings of vertices, where N is the number of vertices and M is the dimension of the initial embedding. The spatial-level GAT is responsible for learning the embedding of vertex *u* based on the initial embedding of *u* and its neighbors *v* in the *t*-th snapshot. Firstly, the spatial-level GAT calculates the importance $\boldsymbol{\alpha}^t[u,v] \in \mathbb{R}$ of *v* to *u* by Eq. (1):

$$\boldsymbol{\alpha}^t = softmax(att_{spa}(\mathbf{A}^t, \mathbf{H}^t)), \tag{1}$$

where $\boldsymbol{\alpha}^t \in \mathbb{R}^{N\times N}$, $softmax(\bullet)$ is the normalized exponential function, $att_{spa}(\bullet)$ is the attention mechanism in spatial-level GAT, $\mathbf{A}^t \in \mathbb{R}^{N\times N}$ is the adjacency matrix of the *t*-th snapshot. Then, it fuses the embeddings of each *v* according to the importance to obtain the spatial embedding $\mathbf{Z}^t[u] \in \mathbb{R}^{1\times F}$ of vertex *u*, shown in Eq. (2).

$$\mathbf{Z}^t = \sigma(\boldsymbol{\alpha}^t \times \mathbf{H}^t), \tag{2}$$

where $\mathbf{Z}^t \in \mathbb{R}^{N\times F}$, F is the dimension of embedding, $\sigma(\bullet)$ denotes the activation function. Finally, the temporal-level GAT calculates the importance of each snapshot to vertex *u* and obtains the final embedding $\mathbf{Z}[u] \in \mathbb{R}^{1\times F}$ of vertex *u* by fusing the embeddings of each snapshot:

$$\mathbf{Z} = \sigma(softmax(att_{tem}(\mathbf{Z}^1, \mathbf{Z}^2, \cdots, \mathbf{Z}^T)) \times \mathbf{Z}^t), \tag{3}$$

where $\mathbf{Z} \in \mathbb{R}^{N\times F}$, $att_{tmp}(\bullet)$ is the attention mechanism in temporal-level GAT. To calculate the attribute similarity between each vertex, we classify all vertices into two categories (category 1 is the vertices with similar attributes to the query vertex, category 2 is the vertices with dissimilar attributes to the query vertex) based on the final embeddings $\mathbf{Z}$, and finally we will obtain the probability that each vertex belongs to category 1, e.g., the attribute similarity. To this end, we use an MLP to calculate the attribute similarity (denote as $\mathbf{AS} \in \mathbb{R}^{N\times 1}$), shown in Eq. (4). To optimize $att_{spa}(\bullet)$, $att_{tmp}(\bullet)$ and MLP, we choose cross-entropy to define the loss function, shown in Eq. (5).

$$\mathbf{AS} = \mathrm{MLP}(\mathbf{Z}), \tag{4}$$

$$loss = \sum_{u\in\mathcal{V}} \left[-u.label \times \log(\mathbf{AS}[u]) - (1-u.label) \times \log(1-\mathbf{AS}[u]) \right], \tag{5}$$

where $u.label$ is the label of vertex *u*, $u.label = 1$ if *u* belongs to the same category as the query vertex, otherwise $u.label = 0$. After all networks converge (the loss function is no longer decreased), the output of MLP is the desired attribute similarity. Of course, this module can also use other GNN based methods to calculate the attribute similarity, such as CS-TGN[12], DySAT[13], TGAT[14], etc.

CS Module. This module aims to complete the process of CS and return the target community C to the front-end in the form of dynamic graph data. In general, C is a

connected graph. In a dynamic graph $\mathcal{G}$, if a vertex has no edges connected to other vertices in any snapshot, then the vertex is an isolated vertex. Thus, if there are no isolated vertices in C, then C is connected. This means that verifying whether C is a connected graph requires traversing all snapshots, which is time-consuming. In order to improve the efficiency, this module will construct a union graph G based on $\mathcal{G}$. The union graph G contains all vertices and edges on all snapshots, it preserves the connectivity between vertices in $\mathcal{G}$. Formally, this module is designed to locate a target community C in a union graph G that meet the following conditions: (1) C contains q and is connected in G; (2) $|C|=s$; (3) $\sum_{q,v\in C}\alpha_{qv}$ is maximum, where q is the query vertex, s is the community size and α_{qv} is the attribute similarity between vertex q and v. To locates the target community, we can use BFS based CS algorithms such as the CS algorithms in ICS-GNN[15], CS-TGN[12] etc.

3. Effectiveness Evaluation

To verify whether our proposed system can search for a community with similar attributes over dynamic graph, we choose three representative baseline methods, including two ACS methods over static graph (BASCS[5] and Greedy-T[15]) and one CS method over dynamic graph (BU-Search[16]). We apply all methods on three real-world datasets (ACM, IMDB and DBLP)[17] and use the accuracy metric (calculated by $\left(\left|V_C \cap V_{C'}\right|\right)/\left(\left|V_C\right|\right)$, where V_C is the set of vertices in the result community, $V_{C'}$ is the set of vertices belonging to the same category as the query vertex) to evaluate the result community C searched by each method.

We implement each algorithm by using Python-3.7.15. For datasets, we divide each dataset into multiple snapshots by the timestamps in it to apply each dataset to the methods over dynamic graphs. For GNN, we set the number of iterations to 200, learning rate to 0.001, and use the Adam optimizer. To avoid the coincidence of experimental results, we randomly use 20 query vertices for experiments and calculate the average value of each metric for each algorithm. Codes of our proposed system are publicly available on GitHub (https://github.com/yixiso/CS-System). The results are shown in Table 1.

Table 1. The accuracy values for different methods

Dataset	BASCS	BU-Search	Greedy-T	Our Method
ACM	0.34674	0.42111	0.53500	**0.99667**
IMDB	0.23438	0.36082	0.51833	**0.81000**
DBLP	0.24833	0.59859	0.60667	**0.99833**

On three datasets, the accuracy values of our method are 57.56%, 64.99% and 75% higher than the best accuracy values of all the baseline methods respectively. This is because the baseline methods are not considering the dynamic changes of attributes, while our method can capture the dynamic changes of attributes, thereby improving the attribute cohesion of the resulting community.

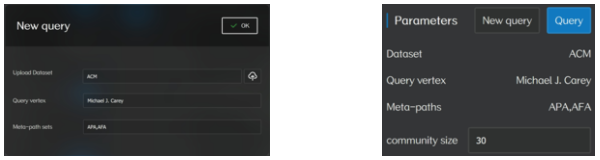
4. Demonstration Scenario

In this section, we illustrate the CS system we proposed in a scenario of recommending similar users. For example, a user named Peter found that he/she was very interested in the research direction and articles written by “Michael J. Carey” when he/she was

reading the literature. So, he/she wanted to find more authors similar to "Michael J. Carey". To this end, the following steps maybe required.

Step 1. Configure Query Parameters. Peter click the "new query" button in the "parameter panel", and then upload the dataset (take ACM as an example), set "Michael J. Carey" as the query vertex, and set the meta-paths to initialize the query (users can leave the meta-paths out to use default parameters) in the pop-up dialog box (Figure 3(a)).

Step 2. Seach and Explore the Community. Peter inputs the community size and click "Query" button (Figure 3(b)) to execute the CS. Then, the result community will be visualized as a static graph in the "Graph Panel" in Figure 2(a) by default. There is a timeline float below the "Graph Panel", and Peter can view each snapshot individually. Each snapshot in this scenario is shown in Figure 4. It can be observed that "Michael J. Carey" has no relations with other authors in snapshot 1 and has many relations in snapshots 2-6. In addition, Peter can also observe which authors "Michael J. Carey" has relations with on each snapshot and discover which authors "Michael J. Carey" has the most frequent relations with. In addition, Peter also can check the details of all vertices and edges in the "Information Panel" as shown in Figure 2(b-c).



(a) The pop-up dialog box (b) The parameters of community search

Figure 3. The pop-up dialog box.

Step 3. Fine-tune the Community. Peter may not be satisfied with the current result community. For example, there are too few/many authors in the result community. Peter can increase/decrease the community size parameter appropriately and click the "Query" button again to re-search the community. The results are shown in Figure 5(a)/(b). Notice that the process of re-searching the community takes very little time. This is because the back-end only need to re-execute the CS module rather than re-execute the whole back-end. So, Peter can adjust the result community easily and quickly. However, if Peter wants to change the query vertex to search for the community, he/she needs to create a new query and re-execute the whole back-end, which means that the efficiency of the system is not ideal for changing the query vertex.

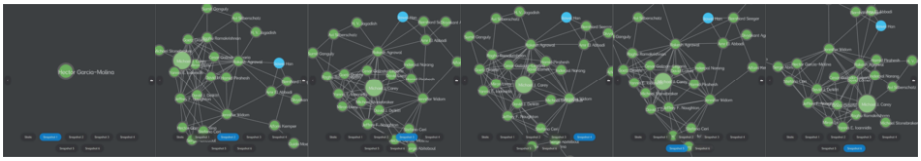
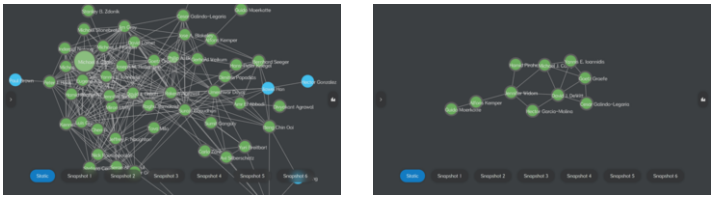


Figure 4. The status of the community at each snapshot.



(a) The community size is 60 (b) The community size is 10

Figure 5. The result community with different community size.

5. Conclusion

We design a CS system over dynamic graph based on graph neural network, making up for the limitation that existing CS systems cannot handle dynamic graphs. Our CS system uses the GNN based methods to capture the dynamic changes of attributes. In addition, it provides users with a friendly interface to conduct CS in dynamic graphs intuitively and it can quickly respond to users' fine-tuning of the results.

Acknowledgments

This research is supported by the National Natural Science Foundation of China (62062066, 62266050 and 62276227), Yunnan Fundamental Research Projects (202201AS070015); the Block-chain and Data Security Governance Engineering Research Center of Yunnan Provincial Department of Education; the Postgraduate Research and Innovation Foundation of Yunnan University (KC-22222861).

References

[1] Y. Fang, X. Huang, L. Qin, Y. Zhang, W. Zhang, R. Cheng, and X. Lin. A survey of community search over big graphs. The VLDB Journal, 2020; 29: 353-392.
[2] Y. Fang, K. Wang, X. Lin, and W. Zhang. Cohesive subgraph search over large heterogeneous information networks. Springer, 2022.
[3] Y. Fang, K. Wang, X. Lin, and W. Zhang. Cohesive subgraph search over big heterogeneous information networks: Applications, challenges, and solutions. In: 2021 International Conference on Management of Data, 2021, pp. 2829-2838.
[4] J. Chen, and J. Gao. VICS-GNN: A visual interactive system for community search via graph neural network. In: 38th International Conference on Data Engineering, 2022, pp. 3150-3153.
[5] J. Wang, L. Zhou, X. Wang, L. Wang, and S. Li. Attribute-sensitive community search over attributed heterogeneous information networks. Expert Systems with Applications, 2024; 235: 121153.
[6] Y. Fang, C. Cheng, S. Luo, and J. Hu. Effective community search for large attributed graphs. Proceedings of the VLDB Endowment, 2016.
[7] X. Huang, and L. V. Lakshmanan. Attribute-driven community search. Proceedings of the VLDB Endowment, 2017; 10(9): 949-960.
[8] B. Liu, F. Zhang, W. Zhang, X. Lin, and Y. Zhang. Efficient community search with size constraint. In: 37th International Conference on Data Engineering, 2021, pp. 97-108.
[9] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. IEEE Transactions on Neural Networks, 2008; 20(1): 61-80.
[10] T. N. Kipf, and M. Welling. Semi-supervised classification with graph convolutional networks. International Conference on Learning Representations, 2017.
[11] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. International Conference on Learning Representations, 2018.
[12] F. Hashemi, A. Behrouz, and M. Rezaei Hajidehi. CS-TGN: Community search via temporal graph neural networks. In: Companion Proceedings of the ACM Web Conference, 2023, pp. 1196-1203.
[13] A. Sankar, Y. Wu, L. Gou, W. Zhang, and H. Yang. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In: 13th International Conference on Web Search and Data Mining, 2020, pp. 519-527.
[14] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan. Inductive representation learning on temporal graphs. arXiv preprint arXiv:2002.07962, 2020.
[15] J. Gao, J. Chen, Z. Li, and J. Zhang. ICS-GNN: Lightweight interactive community search via graph neural network. Proceedings of the VLDB Endowment, 2021; 14(6): 1006-1018.
[16] L. Li, Y. Zhao, Y. Li, F. Wahab, and Z. Wang. The most active community search in large temporal graphs. Knowledge-Based Systems, 2022; 250: 109101.
[17] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu. Heterogeneous graph attention network. In: World Wide Web, 2019, pp. 2022-2032.