2018 IEEE 34th International Conference on Data Engineering

VizCS: Online Searching and Visualizing Communities in Dynamic Graphs

Yuli Jiang *, Xin Huang #, Hong Cheng *, Jeffrey Xu Yu *

* *The Chinese University of Hong Kong,* # *Hong Kong Baptist University*

*{yljiang, hcheng, yu}@se.cuhk.edu.hk, #xinhuang@comp.hkbu.edu.hk

***Abstract*—Given a query vertex in a graph, the task of community search is to find all meaningful communities containing the query vertex in an online manner. In this demonstration, we propose a novel query processing system for searching and visualizing communities in graphs, called VizCS. It exhibits three key innovative features. First, VizCS adopts several community models and supports community search on dynamic graphs where nodes/edges undergo frequently insertions/deletions. Second, VizCS offers a user-friendly visual interface to formulate queries and a real-time response query processing engine. Last but not least, VizCS generates a *community exploration wall* by offering interactive community visualization, which facilitates users to in-depth understanding of the data. Furthermore, VizCS becomes a community search platform that can visualize and compare different community results by various state-of-the-art algorithms and user-uploaded approaches.**

I. Introduction

Communities naturally exist in many real-world networks including social, biological, semantic and communication networks. The task of community detection is to target all communities in a network, which is a fundamental and well-studied problem in the literature [1]. In recent years, community search [2], [3], [4], [5], which aims to find communities containing the query vertex in graphs, has drawn a great deal of attentions. Compared with community detection that finds all communities in the entire network, online community search provides *personalized community discovery* for a query vertex. Because the communities for different vertices in a network may have very different characteristics, this user-centered personalized search is more meaningful [4]. However, online community search faces two key challenges. One challenge is to design a widely used community model for different query vertices. The other challenge is to develop efficient query processing algorithms on large and dynamic graphs.

To model and search the communities of a query vertex, we introduce a concept of ***k-truss community model*** [4]. Given a query vertex q and parameter k, a k-truss community is defined as a maximal subgraph H containing q by satisfying two conditions: (1) every edge is contained in at least $k-2$ triangles in H; (2) every pair of edges can be connected by a series of adjacent triangles in H. Since our truss community model is based on the concept of k-truss, the communities inherit good structural properties of k-truss, such as bounded diameter, k-edge-connected and hierarchical structure [4]. Besides the cohesive structure of discovered communities, the model also has nice properties of few input parameters and efficient query processing. To the best of our knowledge, this is the first demonstration system applying k-truss community model to implement the community search algorithms.

In this demonstration, we present a novel query processing system for searching and visualizing communities in graphs, called VizCS (**Vi**sualiz**z**ation system for **C**ommunity **S**earch). VizCS exhibits the following innovative features. First, it offers a user-friendly visual interface to formulate queries and a powerful query processing engine to efficiently search k-truss communities in an online manner. Second, it generates a *community exploration wall* where the results of discovered communities are depicted in graph visualization and able to interact with users. In addition, *community exploration wall* provides various channels to further search and analyze informative features of community members, which facilitate in-depth understanding of the data. Third, it supports community search over dynamic graphs and provides an interface to upload graph updates with nodes/edges insertions/deletions. Apart from k-truss community model, VizCS platform also supports another community search approach [2] of finding α-adjacency-γ-quasi-k-clique community.

II. System Overview

The system architecture of VizCS is illustrated in Figure 1. It employs a client-server architecture and mainly comprises of the following modules. (1) The *query editor module* interfaces various community models, network data sources and user-raised input queries. (2) The *index maintenance module* consists of two submodules: Index construction and Index update.These two components maintain the index to ensure that VizCS can efficiently work for community search in dynamic graphs. (3) The *query processing module* implements one efficient algorithm that leverages index to find community. (4) The *community exploration wall* uses graph visualization techniques to depict the community results and also presents informative features to users through various exploration channels, such as the profile search of community members by Google, structural statistic report, collaborator recommendation, and tag cloud. In the following, we respectively introduce these modules in detail. To illustrate VizCS more clearly and concretely, following we mainly introduce each modules based on k-truss community model.

2375-026X/18/$31.00 ©2018 IEEE
DOI 10.1109/ICDE.2018.00182

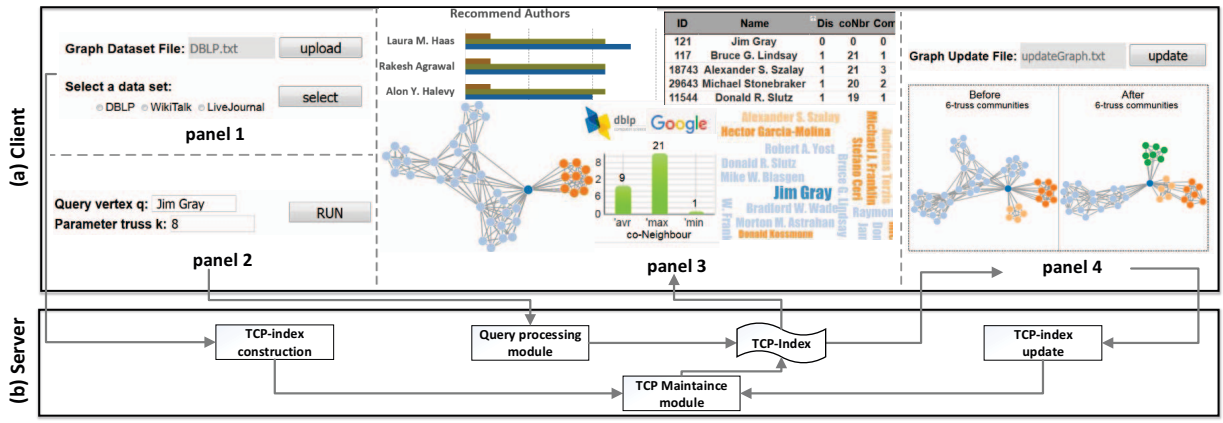


Fig. 1. System Architecture of VizCS

A. Query editor module.

Figure 1 depicts the screenshot of query editor interface of VizCS. It consists of four panels. Panel 1 enables users to select or upload a graph dataset to query. Panel 2 enables users to formulate an input query consisting of a query vertex q and a parameter k. We can execute the query to search all k-truss communities containing q by clicking the "Run" icon. Panel 3 displays the results of communities by graph visualization, and provides various exploration channels. Panel 4 displays the function of community search over dynamic graphs. It enables a user to upload one file of graph updates and compare different results of community search in the original and updated graphs.

B. Index maintenance module.

To offer an efficient search of k-truss communities, this module maintains a compact and elegant index structure called Triangle Connectivity Preserved index (TCP-Index), which supports the search with a linear cost with respect to the community size [4]. Specifically, to support community search over dynamic graphs, this module consists of two components of index construction and index update.

Index construction. Upon selecting a specific community model and graph dataset (e.g., k-truss community and the DBLP network), index construction works to build TCP-Index offline in the server side. The process of index construction is as follows. First, it computes the trussness of each edge of the graph by truss decomposition algorithm [4]. The trussness of an edge is defined as the largest value of k such that there exists k-truss communities containing this edge. Thus, if the trussness of an edge is less than k, the edge will not exist in any k-truss community. Then, to fulfill another constraint of k-truss community model—triangle connectivity, we build the TCP-Index that preserves the trussness value and the triangle adjacency relationship in a tree-shape index. The size of TCP-Index takes the same complexity of graph size, which indicates that it is a very compact index. The time complexity of TCP-Index construction algorithm takes $O(\rho m)$ where ρ is the arboricity of a graph and m is the number of graph edges [4].

Index update. This component performs the update of index in dynamic graphs where nodes and edges can be frequently inserted or deleted. We mainly focus on edge insertion and deletion, because vertex insertion/deletion can be regarded as a sequence of edge insertions/deletions preceded/followed by the insertion/deletion of an isolated vertex. To handle TCP-Index update efficiently for k-truss community, the key is to identify the affected region in the graph precisely. Based on the theoretical analysis shown in [4], this component develops efficient local search algorithms for updating the edge trussness and the TCP-Index respectively.

C. Query processing module.

Based on index maintenance module, this module implements a highly efficient algorithm to process community queries. For k-truss community model [4], given a query vertex q and parameter k, the goal is to find all k-truss communities containing the vertex q in a graph. The algorithm checks every incident edge on vertex q to search k-truss communities. It starts from one incident edge of vertex q with the trussness no less than k, and expands it to search k-truss community in the BFS manner on TCP-Index. This query processing approach returns all k-truss communities in the optimal time complexity, w.r.t. the number of edges in communities. The output of community results is forwarded to the following analysis module.

D. Community exploration wall.

The module generates a community exploration wall to view and analyze the community results in a user-friendly manner. It consists of four following components. We use a collaboration network from the DBLP data set[1] to illustrate. A vertex represents an author and an edge is added between two authors if they have co-authored 3 times or more. The network contains 234,879 vertices and 541,814 edges.

Community visualization. To offer direct, simplified, intuitive and human-friendly images to help users understand the overview of query results, VizCS applies graph visualization tool to depict communities in Figure 2. The visualization tool used in VizCS is the force directed graph[2] , from version 4 of Data-Driven Document(d3)—a JavaScript library.

[1]http://dblp.uni-trier.de/xml/
[2]http://www.puzzlr.org/force-graphs-with-d3/

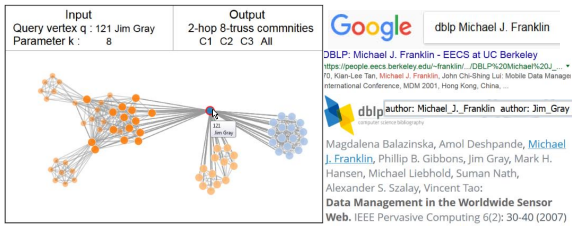


Fig. 2. Community Visualization in *Community Exploration Wall*

VizCS provides several interesting interactive features. First, the nodes from different communities are filled in different colors and user can distinguish them straightforward. Second, the community label (e.g."C_1") in the top middle of Figure 2, stores the information of nodes and edges in this community. If users click the community label, it will trigger the visualization that all nodes in this community will be significantly colored in red. Third, force directed graph provides an interactive layout function that users can click and drag a vertex to adjust the layout of communities. In addition, VizCS embeds the profile information into nodes and edges. Users can click a vertex to see its basic information (e.g. author name and vertex ID) and double click it, which triggers the Google API to search the author's profile further in Figure 2. Also, users can click an edge between two authors to look up their co-authored publications using our embedded API of the DBLP website in Figure 2.

Structural statistic report. This component analyzes the structural statistics of nodes in query communities. Figure 3(a) shows an information table of all authors including the vertex ID, author name, the distance to the query vertex, and the number of common neighbors shared with the query vertex. Users can click one table header to sort all authors in different order. Figure 3(a) presents the authors in the ascending order of vertex ID. Users can also click one table row to view the author in the visualized communities in Figure 2, and double click the row to google search the author for details. In addition, for each community, the distribution statistics of structural information are plotted into bar charts in the top of Figure 3(a). It presents three bar charts of the node degree, the number of common neighbors, and the distance to query node. Each bar chart shows the distribution statistics in terms of its minimum, average, and maximum values. When clicking one community (e.g."C_1") in the top middle of Figure 2, the information of structural statistics will update accordingly for this community.

Collaborator recommendation. This component shows one application of collaborator recommendation, which leverages the results of k-truss community search. Figure 3(b) recommends seven potential collaborators for the query author. The ranking function of recommendation considers three features of the vertex trussness, the distance to query vertex and the number of common neighbors with query vertex. It is an interesting exploration of VizCS, and provides the recommendation results for the manual evaluation by users.

Tag cloud. The component of tag cloud displays authors of query communities using different sized fonts in Figure 3(c). The larger font of one author name is, the closer distance between this author and query author is. In addition, the authors in one community are marked in one color, which is same as the color in Figure 2. Different communities use different colors, which offers direct views. Users can double click the name to search more information about the author in Google.

In summary, the community exploration wall facilitates users to explore results toward their search goals, by various channels of community visualization, structural statistic report, collaborator recommendation, and tag cloud.

III. Related Systems and Novelty

Our work is related with community discovery and graph query processing.

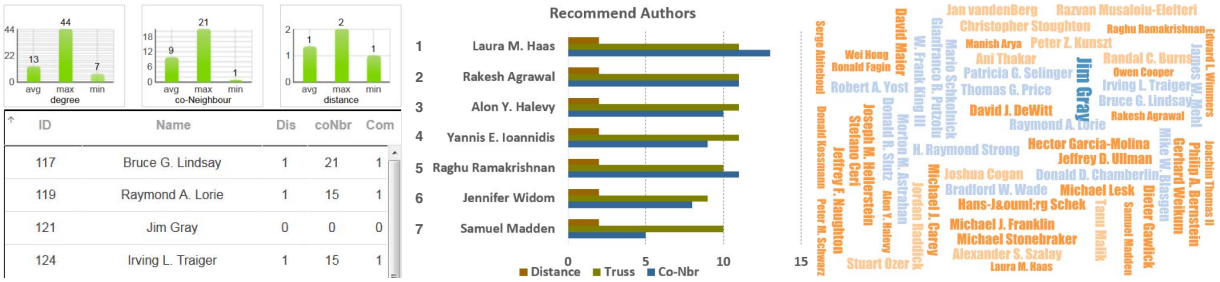
Community discovery. Generally, community discovery can be categorized into community detection and community search. The problem of community detection is to find all communities in the entire network [1]. On the other hand, given a set of query nodes, community search is to find only query-related communities [5]. Several community search models are proposed on various kinds of dense subgraphs such as quasi-clique [2], k-core [5], k-truss [4], and densest subgraph [3]. A brief survey of community search can be found in [7]. Most recently, Fang et al. [8] proposed C-Explorer system to assist user in extracting and analyzing communities equipped with k-core based community search algorithms. In contrast, our system offers k-truss and more community search models and supports online query for dynamic graphs. In addition, SocialLens [9] builds upon community profiling and enables browsing communities by content and interaction. However, this work detects communities offline, which is different from our goal of searching community in an online manner.

Graph query processing. In the literature, there exist several graph systems for handling various kinds of queries. ExpFinder [10] is a graph system for finding experts in social networks based on graph pattern matching. AutoG [11] presents a novel interactive system to alleviate the potentially painstaking task of graph query formulation. GRAPE [12] is a parallel GRAPh query Engine based on a simultaneous fixed point computation in terms of partial and incremental evaluation. All above systems proposed for processing different graph queries on a wide of applications are significantly different from our work on searching and visualizing communities.

IV. Demonstration Overview

We implement the community search algorithms of VizCS in C++, and develop webpages to access server. The key objectives of the demonstration are to enable the audience to interactively experience the following.

User-friendly input interfaces. With the query editor module and query processing module, audiences can search communities in the light of their inputs. Since the query communities may consist of a large number of nodes and edges, it is hard



(a) Structural Statistic Report (b) Collaborator Recommendation (c) Tag Cloud

Fig. 3. Community Exploration Wall

to overview the results if we directly visualize all nodes and edges in the query communities. To make the complexity of displayed results within human cognitive capacity, we depict a limited number of community members in a near-vicinity of the query vertex. In addition, If audiences raise a query only containing one query vertex without the input of value k, VizCS can automatically construct one complete query that sets the value of k as the largest k-truss containing the query vertex.

Interactive community exploration wall. Audiences can view and interact with results in the community exploration wall. Specifically, it displays 4 userful visualization panels. First, all communities are depicted in graph visualization in Figure 2. Different communities are marked with different colors. Audiences can click nodes and edges to gain more information and drag nodes to better undertand communities structure. Second, the statistics of various structural information in communities are listed in tables and plotted in bar charts in Figure 3(a). Users can click the grid headers to sort the list of authors in different order, e.g., in the ascending order of vertex ID. Third, VizCS recommends a list of potential collaborators for the query author in Figure 3(b). Last but not least, VizCS further visualizes the community members using one tag cloud in Figure 3(c). The larger font of one author name is, the closer distance between this author and query author has, which offers a straightfoward view.

Community search over dynamic graphs. Upon uploading one file of graph updates, audiences can explore the feature of community search over dynamic graphs. For k-truss community model, TCP-Index can be incrementally updated by the index maintenance module of VizCS. Audiences can compare different results for one query in envolving graphs in Figure 4.

Comparison of state-of-the-art methods. VizCS can load other state-of-the-art community search approaches into VizCS, and compare the difference of community results by various methods. There are several different dense subgraph based community models including quasi-clique [2], the denset subgraph [3], and k-core [5]. Our demonstration framework can equip with the implementations of other community search engines and visualize community results for audiences to understand the difference models vividly and directly. In addition, VizCS can serve as a platform that provides an interface

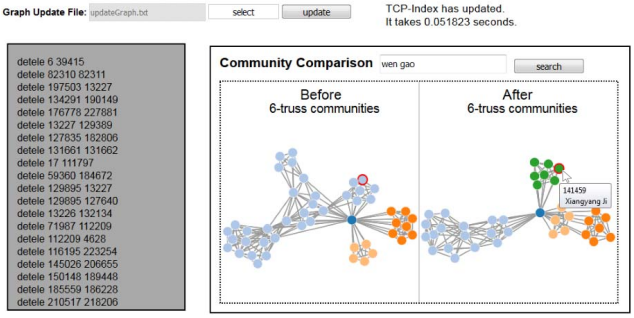


Fig. 4. Community search over dynamic graphs

for uploading user-customized community search result and viewing its visualization.

ACKNOWLEDGMENT

This work was supported by grants from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No.: CUHK 14205617], [Project No.: CUHK 14221716], and [Project No.: HKBU 12200917].

REFERENCES

[1] M. E. Newman, "Fast algorithm for detecting community structure in networks," *Physical review E*, vol. 69, no. 6, p. 066133, 2004.
[2] W. Cui, Y. Xiao, H. Wang, Y. Lu, and W. Wang, "Online search of overlapping communities," in *SIGMOD*, 2013, pp. 277–288.
[3] Y. Wu, R. Jin, J. Li, and X. Zhang, "Robust local community detection: on free rider effect and its elimination," *PVLDB*, vol. 8, no. 7, pp. 798–809, 2015.
[4] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu, "Querying k-truss community in large and dynamic graphs," in *SIGMOD*, 2014, pp. 1311–1322.
[5] M. Sozio and A. Gionis, "The community-search problem and how to plan a successful cocktail party," in *KDD*, 2010, pp. 939–948.
[6] X. Huang, L. V. Lakshmanan, J. X. Yu, and H. Cheng, "Approximate closest community search in networks," *Proceedings of the VLDB Endowment*, vol. 9, no. 4, pp. 276–287, 2015.
[7] X. Huang, L. V. Lakshmanan, and J. Xu, "Community search over big graphs: Models, algorithms, and opportunities," in *ICDE*, 2017, pp. 1451–1454.
[8] Y. Fang, R. Cheng, S. Luo, J. Hu, and K. Huang, "C-explorer: browsing communities in large graphs," *PVLDB*, vol. 10, no. 12, 2017.
[9] H. Cai, V. W. Zheng, P. Chen, F. Zhu, K. C.-C. Chang, and Z. Huang, "Sociallens: Searching and browsing communities by content and interaction," in *ICDE*, 2017, pp. 1397–1398.
[10] W. fei Fan, X. Wang, and Y. Wu, "Expfinder: Finding experts by graph pattern matching," in *ICDE*, 2013, pp. 1316–1319.
[11] P. Yi, B. Choi, S. S. Bhowmick, and J. Xu, "Autog: A visual query autocompletion framework for graph databases," *PVLDB*, vol. 9, no. 13, pp. 1505–1508, 2016.
[12] W. Fan, J. Xu, Y. Wu, W. Yu, and J. Jiang, "Grape: parallelizing sequential graph computations," *PVLDB*, vol. 10, no. 12, pp. 1889–1892, 2017.