C-Explorer: Browsing Communities in Large Graphs

Yixiang Fang, Reynold Cheng, Siqiang Luo, Jiafeng Hu, Kai Huang
Department of Computer Science, The University of Hong Kong, Pokfulam Road, Hong Kong
{yxfang, ckcheng, sqluo, jhu, khuang}@cs.hku.hk

ABSTRACT

Community retrieval (CR) algorithms, which enable the extraction of subgraphs from large social networks (e.g., Facebook and Twitter), have attracted tremendous interest. Various CR solutions, such as k-core and CODICIL, have been proposed to obtain graphs whose vertices are closely related. In this paper, we propose the *C-Explorer* system to assist users in extracting, visualizing, and analyzing communities. *C-Explorer* provides online and interactive CR facilities, allowing a user to view her interesting graphs, indicate her required vertex q, and display the communities to which q belongs. A seminal feature of *C-Explorer* is that it uses an *attributed graph*, whose vertices are associated with labels and keywords, and looks for an *attributed community* (or AC), whose vertices are structurally and semantically related. Moreover, *C-Explorer* implements several state-of-the-art CR algorithms, as well as functions for analyzing their effectiveness. We plan to make *C-Explorer* an open-source web-based platform, and design API functions for software developers to test their CR algorithms in our system.

1. INTRODUCTION

Gigantic social networks are prevalent in important and emerging applications such as Facebook, Twitter, and Meetup. In these systems, a fundamental task is *community retrieval* (CR), a process of extracting groups of social network users who have close relationship. Communities, which reflect common interests and behaviors among users, can be useful for many real applications such as setting up social events [11, 4] and designing promotion strategies [13]. Communities are also interesting to social scientists and historians for analysis purposes [7]. Due to the importance of finding communities, plenty of CR algorithms have been proposed in recent years (e.g., [11, 1, 7, 4, 3, 6, 9, 10]).

In this paper, we propose **C**ommunity **E**xplorer (or *C-Explorer*), a web-based system that enables CR in a simple, online, and interactive manner. Figure 1 shows the user interface of *C-Explorer* configured to run on the DBLP bibliographical network. On the left panel, a user inputs the name of an author (e.g., "jim gray") and the minimum degree of each vertex in the community she wants to have. The user can also indicate the labels or *keywords* related to her community. Once she clicks the "Search" button, the right panel will quickly display a community of Jim Gray, which contains researchers working on database system transactions. The user can further click on one of the vertices (e.g., Michael Stonebraker), after which the profile related to the Michael will be shown (Figure 2). The user can continue to examine Michael's community.

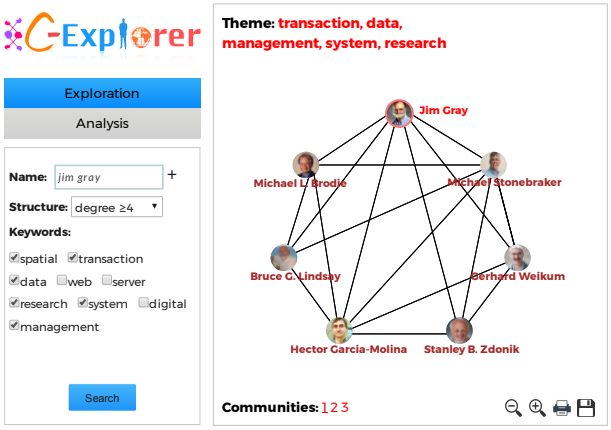


Figure 1: Interface of *C-Explorer*.

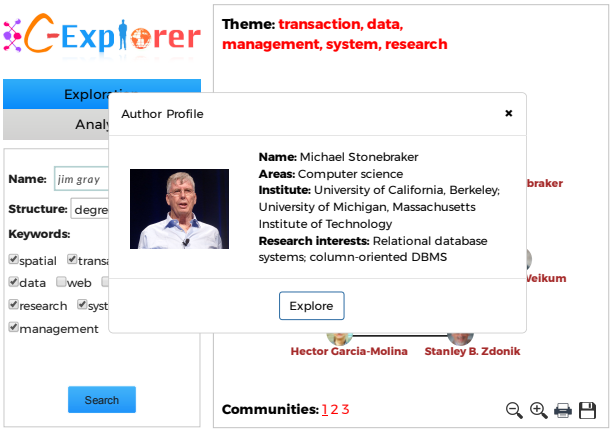


Figure 2: The profile of a community member.

Let us now discuss three key features of *C-Explorer*.

- **Attributed graphs and communities**. The CR solution employed by *C-Explorer* involves the use of the keyword information associated with the vertices in the social network. As illustrated in Figure 1, the keywords displayed in the left panel are extracted

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org.
Proceedings of the VLDB Endowment, Vol. 10, No. 12
Copyright 2017 VLDB Endowment 2150-8097/17/08.

from Jim Gray's articles in DBLP, and they are also used to define the "theme" of the community (right panel), whose vertices contain keywords related to the theme. We call a graph tagged with keywords an *attributed graph* [13], and name the community computed from the graph structure and keywords an *attributed community*, or *AC* [4]. The problem of finding ACs over an attributed graph has only been recently studied. Also, as shown in [4], ACs are experimentally better than communities generated by other solutions. Moreover, we are also not aware of any other systems that provide CS functionalities over attributed graphs.

• **Support for other CR algorithms**. Besides AC search algorithms, *C-Explorer* supports state-of-the-art CR algorithms, including `Global` [11], `Local` [1], and `CODICIL` [10]. A user can also plug in her own CR solution on *C-Explorer* through a simple application programmer interface (API). The API has a list of functions for further development. Furthermore, *C-Explorer* is an open-source software, and so application developers can customize *C-Explorer* to suit their needs.

• **Analysis facilities**. Our system provides a user-friendly facility that enables online visualization of communities generated by different algorithms, as well as statistics about the communities created (e.g., the number of vertices constituting the community, and the average vertex degree). The measures quantifying the quality of the communities can be displayed in charts. Moreover, the communities generated by different solutions can be viewed simultaneously. These features allow users to perform a detailed comparison on the communities output by different algorithms.

Why do we design *C-Explorer* in this way? Although many CR algorithms have been proposed in the literature, it is not entirely clear which one is better. In previous works, experimental comparison was often performed among several algorithms on a few datasets. Our system enables a more extensive experimental evaluation of CR solutions on a variety of datasets. Moreover, there lacks tools for users to pick the right CR algorithm, analyze communities, or use the solution in their own applications. As discussed above, *C-Explorer* will incorporate an API that facilitates installation and testing of new CR solutions, as well as invoking of CR solutions in the application. Our system will be valuable to users who are interested in performing CR tasks (e.g., database researchers, application developers, social scientists).

The rest of the paper is organized as follows. In Section 2, we discuss existing solutions and the novelty of *C-Explorer*. Section 3 describes the framework and detailed design of *C-Explorer*. In Section 4 we explain how we are going to demonstrate *C-Explorer*.

2. RELATED WORK AND NOVELTY

Community retrieval. There are two classes of CR methods, namely community detection (CD) and community search (CS). In general, CD algorithms aim to retrieve all communities for a graph [9, 5, 10]. These solutions are not "query based", i.e., they are not customized for a query request (e.g., a user-specified query vertex). Moreover, they may take a long time to find all the communities for a large graph, and so they are not suitable for quick or online retrieval of communities. To solve these problems, CS solutions have been recently developed [11, 1, 7].

CS solutions are often query-based (e.g., a user specifies the vertex for which the community is retrieved), and derive communities in an "online" manner. To measure the structure cohesiveness of a community, the *minimum degree* is often used [11, 1]. Sozio et al. [11] proposed the first algorithm `Global` to find the k-$\widehat{core}$ containing q. Cui et al. [1] proposed `Local`, which uses local expansion techniques to enhance the performance of `Global`. Other structure cohesiveness measures, including connectivity [6] and k-truss [7], have also been considered for searching communities. Recently, we have proposed a CS solution, called ACQ that produces communities on attributed graphs [4]. We use ACQ in the engine of *C-Explorer*.

Although plenty of CR solutions have been developed, often they were evaluated on few datasets. Due to the lack of a software tool for researchers to perform a detailed analysis of communities, we build *C-Explorer*, which contains an API for installing new CR tools, as well as functions to compare different CR algorithms.

Graph query tools. In [12], Spillane et al. developed a system G*, which stores graphs on a large number of servers and allows users to easily express graph queries. In [2], Fan et al. presented ExpFinder, a system for finding experts in social networks based on graph pattern matching. In [8], a system called VIIQ was presented, which enables a user interface for interactive graph query formulation. In [14], Yi et al. presented another system called AutoG, which is able to return the top-k graph query suggestions at interactive time.

The above systems, designed for general graph queries, are not customized for browsing of communities. However, it is not straightforward to implement CR algorithms by simple graph queries. We will develop *C-Explorer* to enable online exploration, and visualization and analysis of communities.

3. SYSTEM OVERVIEW

Figure 3 illustrates the system framework of *C-Explorer*, which adopts the browser-server model. The *Browser* side provides interfaces for users to submit queries, view the returned communities, and observe the analysis results. To issue a query, a user enters the name of query vertex and input its parameter values via the browser. The query is sent to the server for processing, after which the communities are displayed on the browser.

The *Server* side has two modules, namely i.e., *Community Search* and *Comparison Analysis*, used for answering queries and performing comparison analysis online respectively. To facilitate efficient community search, we build an index in the *Indexing* module. The community exploration is based on the ACQ query [4], which we will detail in Section 3.2.

The *Comparison Analysis* module performs similarity analysis for the communities found by different methods. Furthermore, it can report the statistics (e.g., the numbers of vertices and edges) of communities retrieved by different CR algorithms. In addition, our system allows other CR algorithms to be plugged in the *Other CR Algorithm* module. We provide a list of Java API functions, so the public users can easily plug in their own algorithms, upload their own graphs, and perform community exploration and comparison analysis. More details about the API functions are discussed in Section 3.1. Currently, we have implemented two other CS algorithms, i.e., `Global` [11] and `Local` [1], and one CD algorithm `CODICIL` [10]. Note that `Global` [11] and `Local` [1] are based on the concept of k-core. We remark that *C-Explorer* employs modular design, which facilitates future extension.

3.1 API

Our system provides several Java interfaces, which consist of a list of API functions. For public users, to plug in their own CR methods, they just need to implement the functions in the interfaces with their own CR algorithms, and slightly modify the webpage codes. Then they can visualize and analyze the communities retrieved by different methods in *C-Explorer*. Figure 4 depicts five typical functions in the main interface, which are illustrated as follows.

• `upload`: it uploads a local graph into the system.

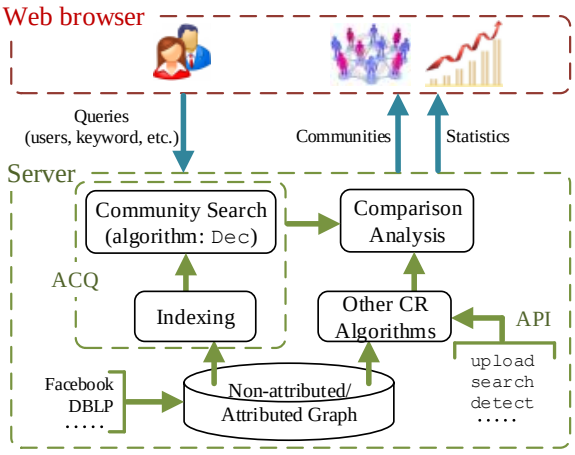


Figure 3: The framework of *C-Explorer*.

• `search`: it searches the communities of a query vertex with some parameters using a specific CS algorithm.
• `detect`: it detects all the communities from an entire graph with a given CD algorithm.
• `analyze`: it performs analysis (e.g., computing similarities of vertices) on the communities with a specific metric.
• `display`: it computes the layout (i.e., locations of vertices and edges) of a given community in a plane, which will be used for visualization in the browser.

```
public interface CExplorer {
    ...
    public void upload(String filePath);
    public List<Community> search(CSAlgorithm algo, Query query);
    public List<Community> detect(CDAlgorithm algo);
    public void analyze(Community community);
    public void display(Community community);
    ...
}
```

Figure 4: A list of API functions.

3.2 ACQ Query

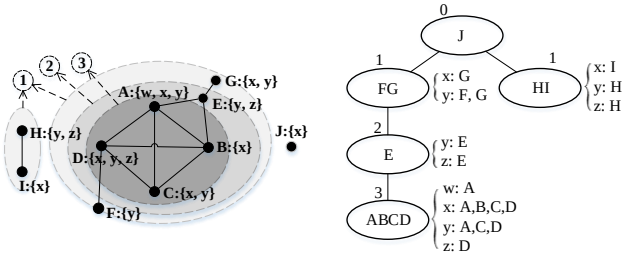
Problem. We consider an undirected attributed graph $G(V, E)$ with vertex set V and edge set E. Each vertex $v \in V$ is associated with a set, $W(v)$, of keywords, and its degree in G is denoted by $deg_G(v)$. The definition of ACQ Query [4] is as follows.

PROBLEM 1 (ACQ). *Given a graph $G(V, E)$, a positive integer k, a vertex $q \in V$ and a set of keywords $S \subseteq W(q)$, return a set $\mathcal{G}$ of graphs, such that $\forall G_q \in \mathcal{G}$, the following properties hold:*

- ***Connectivity***. *$G_q \subseteq G$ is connected and contains q;*
- ***Structure cohesiveness***. *$\forall v \in G_q$, $deg_{G_q}(v) \geq k$;*
- ***Keyword cohesiveness***. *The size of $L(G_q, S)$ is maximal, where $L(G_q, S) = \cap_{v \in G_q}(W(v) \cap S)$ is the set of keywords shared in S by all vertices of G_q.*

Let us take the attributed graph in Figure 5(a) as an example, where there are 10 vertices $\{A, B, \cdots, J\}$ and 11 edges. Each vertex has at most three keywords. If q=A, k=2 and S=$\{w, x, y\}$, then the output of the ACQ query is the subgraph of three vertices $\{A, C, D\}$, and all the vertices share two keywords x and y.

In addition, we develop a variant of the ACQ query supporting multiple query vertices. Specifically, given $G(V, E)$, k, S, and a set Q of query vertices, it returns a set $\mathcal{G}$ of graphs, such that



(a) An example graph (b) A CL-tree index

Figure 5: An example graph and its CL-tree index [4].

$\forall G_q \in \mathcal{G}$, G_q is a connected subgraph containing Q, and satisfies the structure and keyword cohesiveness in Problem 1.

A straightforward method to answer the ACQ query is first to consider all the possible keyword combinations of S, and then return the subgraphs, which satisfy the minimum degree constraint and have the most shared keywords. This method requires the enumeration of all the subsets of S, which implies that it has a complexity exponential to the size of S. Thus it is impractical, especially when there are many keywords in S. To enable efficient retrieval of ACs, we propose a CL-tree index [4].

CL-tree index. The CL-tree organizes all the k-cores and keywords in a tree structure, where the k-core, H_k, is the largest subgraph of the graph G, such that for any vertex in H_k, its degree is at least k ($k \geq 0$). Note that H_k may not be connected. In Figure 5(a), each connected component of a k-core is depicted in an ellipse. The CL-tree is built based on the key observation that k-cores are nested, i.e., a $(k$+1$)$-core must be contained in a k-core. Figure 5(b) depicts a CL-tree for the attributed graph in Figure 5(a).

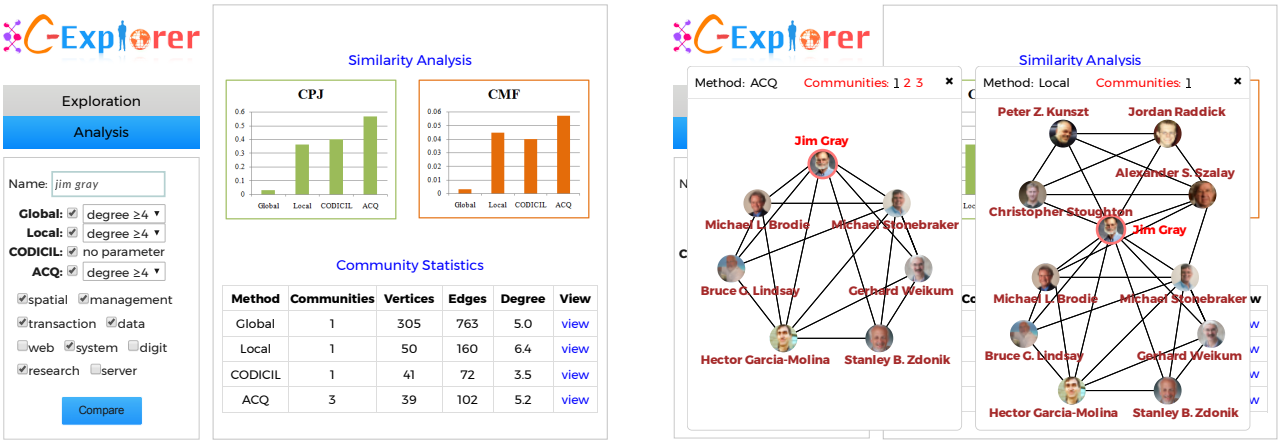
In the CL-tree, each node [1] contains some vertices of the graph. The subtree rooted at each node represents a connected component of the k-core. For each keyword w that appears in a CL-tree node, a list of IDs of vertices whose keyword sets contain w is stored. The CL-tree allows us to locate a specific k-core and verify whether a k-core contains a particular keyword or not efficiently. Thus, it enables the ACs to be found efficiently. It is worth noting that, the CL-tree can be built in linear space and time cost.

Query algorithms. Based on the CL-tree index, we propose three efficient query algorithms. Considering the methods of verifying whether a keyword combination result in an AC or not, we divide the query algorithms into incremental algorithms (from examining smaller candidate sets to larger ones) and decremental algorithm (from examining larger candidate sets to smaller ones) [4]. The incremental algorithms are called `Inc-S` and `Inc-T`, while the decremental algorithm is called `Dec`. Since `Dec` is generally faster than `Inc-S` and `Inc-T`, we choose `Dec` for the system.

4. DEMONSTRATION

Setup and dataset. We implement all the CR methods in Java and design the website using JavaServer Pages (JSP) technique with the Tomcat server. We use a graph sampled from the DBLP bibliographical network. The graph contains 977,288 vertices and 3,432,273 edges. Each vertex denotes an author, and each edge is a co-authorship relationship between two authors. For each author, we use the 20 most frequent keywords in the titles of her publications as her keywords. We also extract the profiles of several hundreds of renowned researchers in the database area from Wikipedia.

[1]We use "node" to mean "CL-tree node" in this paper.



(a) Similarity and statistic comparison

(b) Visual comparison

Figure 6: Comparison analysis.

Community exploration. In this scenario, users can perform community exploration, as shown in Figure 1. First of all, users should specify the name of the query author. Note that by clicking the icon "+" on the right-hand side of the input box, users can input more than one query author. After that, the system will display the structure constraints, i.e., a list of degree constraints, and a set of keywords of this author. Then, users can click the "Submit" button to issue an ACQ query after selecting these parameters. Next, the server receives the parameters and performs ACQ query to obtain the communities. Finally, the communities will be returned instantly and displayed in the browser. Moreover, to have a clearer view on some vertices, users can zoom in or zoom out the figure of the community by clicking the corresponding icons. In addition, users can save the community into a .jpg file or print it directly.

Before further exploring the communities of a vertex, users can simply click the portrait of the researcher. Then, the profile of the author is shown in a new window. In Figure 2, the profile of Michael Stonebraker is depicted. Users can then continue to explore Michael's communities. Note that for the layout of vertices of the community, we use the algorithms of the JUNG project [2].

Comparison analysis. Our system allows query users to compare the communities retrieved by various CR algorithms, in terms of community quality and statistics. For community quality, we mainly evaluate the vertices' similarities. In [4], we propose two metrics: CPJ and CMF. The metric CPJ measures the average similarity over all pairs of vertices, and the metric CMF measures the average frequency of keywords in $W(q)$ for all the vertices in the community. In general, the higher values of CPJ and CMF imply better cohesiveness of a community.

By analyzing the similarities of vertices in the communities and showing their CMF and CPJ values, the user can easily compare different CR methods. For statistics, the system can display the numbers of returned communities, as well as their average numbers of vertices, edges, and degrees. Also, the links for visualizing these communities are given, and the user can view them simultaneously.

In Figure 6(a), the user can get into the analysis interface by clicking the "Analysis" link on the top of left panel. Then, the user can input the name of query author (e.g., Jim Gray), select the algorithms, and input their parameters on the left panel. After clicking the "Compare" button, the CPJ and CMF values of communities retrieved by different methods are depicted in bar graphs on the right panel. Also, the statistics of these communities are reported in the table below. Moreover, the user can click the "view" links in the table to visualize the communities in new windows. In Figure 6(b), two communities found by ACQ and `Local` are presented, and their differences can be easily observed.

[2]http://jung.sourceforge.net/

Acknowledgments

Yixiang Fang, Reynold Cheng, Siqiang Luo, Jiafeng Hu, and Kai Huang were supported by the Research Grants Council of Hong Kong (RGC Projects HKU 17229116 and 17205115) and the University of Hong Kong (Projects 102009508 and 104004129). We also thank the reviewers for their insightful comments.

5. REFERENCES

[1] W. Cui, Y. Xiao, H. Wang, and W. Wang. Local search of communities in large graphs. In *SIGMOD*, pages 991–1002, 2014.

[2] W. Fan, X. Wang, and Y. Wu. Expfinder: Finding experts by graph pattern matching. In *ICDE*, pages 1316–1319. IEEE, 2013.

[3] Y. Fang, R. Cheng, X. Li, S. Luo, and J. Hu. Effective community search over large spatial graphs. *PVLDB*, 10(6):709–720, 2017.

[4] Y. Fang, R. Cheng, S. Luo, and J. Hu. Effective community search for large attributed graphs. *PVLDB*, 9(12):1233–1244, Aug. 2016.

[5] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

[6] J. Hu, X. Wu, R. Cheng, S. Luo, and Y. Fang. Querying minimal steiner maximum-connected subgraphs in large graphs. In *CIKM*, pages 1241–1250. ACM, 2016.

[7] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. In *SIGMOD*, pages 1311–1322. ACM, 2014.

[8] N. Jayaram, S. Goyal, and C. Li. Viiq: auto-suggestion enabled visual interface for interactive graph query formulation. *PVLDB*, 8(12):1940–1943, 2015.

[9] M. E. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

[10] Y. Ruan, D. Fuhry, and S. Parthasarathy. Efficient community detection in large networks using content and links. In *WWW*, pages 1089–1098. ACM, 2013.

[11] M. Sozio and A. Gionis. The community-search problem and how to plan a successful cocktail party. In *SIGKDD*, pages 939–948, 2010.

[12] S. R. Spillane et al. A demonstration of the g? graph database system. In *ICDE*, pages 1356–1359. IEEE, 2013.

[13] Z. Xu, Y. Ke, Y. Wang, H. Cheng, and J. Cheng. A model-based approach to attributed graph clustering. In *SIGMOD*, pages 505–516. ACM, 2012.

[14] P. Yi, B. Choi, S. S. Bhowmick, and J. Xu. Autog: A visual query autocompletion framework for graph databases. *PVLDB*, 9(13):1505–1508, 2016.