

M.SC. ENGG. THESIS

A SCALABLE ALGORITHM FOR KEYWORD
AWARE INFLUENTIAL COMMUNITY SEARCH
IN LARGE SOCIAL NETWORKS

by

Md. Saiful Islam

Submitted to

Department of Computer Science and Engineering

in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science and Engineering



Department of Computer Science and Engineering

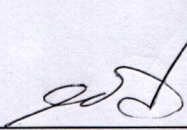
Bangladesh University of Engineering and Technology (BUET)

Dhaka 1000

18 August, 2020

The thesis titled "A Scalable Algorithm for Keyword Aware Influential Community Search in Large Social Networks", submitted by Md. Saiful Islam, Roll No. **0417052006 P**, Session April 2017, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering and approved as to its style and contents. Examination held on 18 August, 2020.

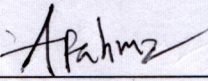
Board of Examiners

1.  _____

Dr. Mohammed Eunus Ali
Professor

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology, Dhaka.

Chairman
(Supervisor)

2.  _____

Dr. A.K.M. Ashikur Rahman
Head and Professor

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology, Dhaka.

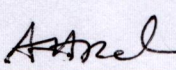
Member
(Ex-Officio)

3.  _____

Dr. Md. Saidur Rahman
Professor

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology, Dhaka.

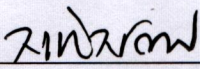
Member

4.  _____

Dr. Atif Hasan Rahman
Assistant Professor

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology, Dhaka.

Member

5.  _____

Dr. Salekul Islam
Professor

Department of Computer Science and Engineering (CSE)
United International University (UIU), Dhaka.

Member
(External)

Candidate's Declaration

This is hereby declared that the work titled "A Scalable Algorithm for Keyword Aware Influential Community Search in Large Social Networks" is the outcome of research carried out by me under the supervision of Dr. Mohammed Eunos Ali, in the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka 1000. It is also declared that this thesis or any part of it has not been submitted elsewhere for the award of any degree or diploma.

Saiful
18-08-2020

Md. Saiful Islam

Candidate

Acknowledgment

I am thankful to my supervisor, Dr. Mohammed Eunos Ali, for his constant supervision and guidance. I express my heartiest gratitude to Dr. Yong-Bin Kang, Monash University, Australia for his keen interest and collaboration throughout the study. I am grateful to Swinburne University, Australia for providing access to their supercomputer, without which carrying out the study would have been very difficult.

I would also want to thank the members of my thesis committee for their valuable suggestions. I thank Dr. A.K.M. Ashikur Rahman, Dr. Md. Saidur Rahman, Dr. Atif Hasan Rahman, and especially the external member Dr. Salekul Islam.

I am especially grateful to the Department of Computer Science and Engineering (CSE) of Bangladesh University of Engineering and Technology (BUET) for providing their support during the thesis work. I sincerely thank CSE Office staff for providing logistic support to me to successfully complete the thesis work.

I would like to thank my family, my friends, and all of those who supported me with their appreciable assistance, patience, and suggestions during the course of my thesis. Finally, I express my gratitude to the Almighty for keeping me in good health.

Abstract

An *influential community* is defined as a closely connected group of people who have some dominance over the populace. In a social network, usually each individual has expertise in various topics. We consider the scenario where the network is represented as an *attributed graph*, users being the vertices, and their social connections being the edges. The attributes of each node is a set of keywords, representing the topics in which the corresponding user has some expertise. Additionally, for each user, keywords are paired with an expertise score representing how much expertise the user has in the representative topic. In such an attributed graph, we study the problem of finding the most influential communities given a combination of keywords as a query. A concern in keyword based community search is that, there can be millions of keywords in real life social networks. It is not user friendly to assume that users can raise queries using the keywords exactly as in the attributed graph. In this context, we propose a novel word-embedding based similarity model that enables *semantic community search*, which substantially alleviates the limitations of *exact keyword* based community search. Next, we propose a new influence measure for a community that considers both the cohesiveness of the community and the expertise scores of the members of the community in topics relevant to the query. Such a measure eliminates the need for specifying values of internal parameters of a network. Finally, we propose two efficient algorithms followed by a basic solution for searching influential communities in large attributed graphs. We present detailed experiments and a case study to demonstrate the effectiveness and efficiency of the proposed approaches.

Contents

<i>Board of Examiners</i>	1
<i>Candidate's Declaration</i>	2
<i>Acknowledgment</i>	3
<i>Abstract</i>	4
Contents	5
List of Figures	8
List of Tables	9
1 Introduction	1
1.1 Motivation and Applications	2
1.2 State of the Art and Research Gaps	3
1.3 Challenges and Contributions	5
1.4 Organization	7
2 Related Works	8
2.1 Keyword search	8
2.2 Cohesive subgraph mining	9
2.3 Community detection	9
2.4 Team formation in social networks	9

2.5	Community search	10
3	Problem Definition and System Overview	12
3.1	Problem definition	12
3.2	System overview	14
4	Constructing Attributed Graph	17
4.1	Semantic keyword similarity model	17
4.2	Constructing attributed graph	20
4.3	Augmenting keywords for KICQ	20
5	Influential Community Measures	21
5.1	Query relevance score of a vertex	21
5.2	Desiderata of an influential community	22
5.3	Influential score function	23
6	Algorithms for Influential Community Search	24
6.1	A straightforward approach, BASIC-EXPLORE	24
6.1.1	Finding query essential subgraph	24
6.1.2	Finding k -cores and most influential communities	25
6.1.3	Time complexity	26
6.2	Pruned exploration approach, PRUNED-EXPLORE	27
6.3	Keyword indexed tree exploration, TREE-EXPLORE	30
6.3.1	KIC-tree index	31
6.3.2	Complexity analysis for index construction	33
6.3.3	Computing upper bound scores for a query	34
6.3.4	TREE-EXPLORE algorithm	35
7	Experimental Study	38
7.1	Experimental setup	38
7.1.1	Datasets	39

7.1.2	Query Generation	39
7.1.3	Experiment parameters	40
7.2	Evaluation of semantic similarity model	41
7.2.1	Selection of similarity metric	41
7.2.2	Parameter setting	43
7.3	Evaluation of KICQ	45
7.3.1	Performance evaluation	45
7.3.2	Comparison with state-of-the-art	50
7.4	Experiments with Gowalla dataset	51
8	Case Study	53
9	Conclusion	56
	Bibliography	57

List of Figures

1.1	An attributed author-author graph, where each vertex has an associated list of attributes (keywords) and influences denoting her expertise. Different types of CS communities are marked as $C_1 - C_4$.	4
3.1	The overview of the proposed system	15
6.1	A query essential subgraph.	26
6.2	A subgraph of the graph presented in Figure 1.1.	31
6.3	KIC-tree for the attributed graph in Figure 6.2	32
7.1	$L=15$ (L : number of top words) provides reasonably high keyword coherence, low Davies-Bouldin index, and high retrieval percentage	44
7.2	Query processing time for varying dataset size	46
7.3	Query processing time for varying k_{min}	47
7.4	Query processing time for varying r	48
7.5	Index size for varying vertices and keywords	48
7.6	Performance measures for varying β values	50
7.7	Effectiveness and efficiency comparison	51
7.8	Effectiveness and efficiency comparison	52
8.1	Retrieved top communities for DS.	53
8.2	Top communities for BN OR DM.	54

List of Tables

2.1	Existing works on community search	10
7.1	Parameters for experimental analysis	40
7.2	Performance results of the three similarity metrics	43
7.3	Structural cohesiveness measures	49
8.1	Top communities by our approach in DS.	55

Chapter 1

Introduction

Community is generally defined as a group of individuals who have close interaction among themselves. Communities serve as a basic structure for understanding the organization of many real-world networks or graphs. These networks include academic networks like DBLP, social networks like Facebook or Twitter, biological networks like protein-protein interactions, and many more. These networks are usually modelled by graphs, where vertices represent the individuals and edges represent the interaction between individuals. Finding communities from such large graphs has received significant attention in recent years due to its diverse practical applications that include event organization [1], friend recommendation [2], and e-commerce advertisement [3]. Traditionally, community search (CS) on a large graph involves finding a community around a given query vertex that satisfies *query parameters* like *connectivity* and *cohesiveness constraints* [1,4–6]. For example, by using such techniques, one can find a community from the DBLP network for an author as a query, where the community should be a connected subgraph and each member should be connected to at least two other members in the community.

More recent research works [7, 8] have focused on *finding influential communities* from a graph. The common goal of finding influential communities is to find a closely connected group of users (vertices) who have some dominance over other users in the graph.

We consider a social network where the members have expertise in several keywords. In this study, we address the problem of finding influential communities based on a particular set of keywords that matches with the interest of the user.

The rest of the chapter is organized as follows. We briefly describe the motivations and applications of this study in Section 1.1. We discuss the research gaps in the state-of-the-art literature in Section 1.2. Then, we present the challenges of this study and highlight our contributions in Section 1.3. Finally, we present how the remaining chapters are organized in Section 1.4.

1.1 Motivation and Applications

An influential community is a group of highly influential individuals who can affect the emotions, opinions, or behavior of other people in the network. However, the interpretation of influence depends varies from person to person. A community where the members are expert in music will be highly influential for a person interested in music, while, for a person who does not like music, the same community may not be influential at all.

Some existing works [4, 6] address the problem of topic based community search, but they require a set of query vertex from the user. In a social network, a user may want to find communities of specific interest without mentioning any query vertex or internal graph properties. For example, a new researcher may want to find top research groups in topics of her interest while applying for PhD admission. To her, a query like, “find top research groups in machine learning” is easier to raise than queries like “find top research groups including Yoshua Bengio.”

Searching communities with specific expertise (e.g., “machine learning”) is beneficial for not only academic network, but also for social networks (e.g., Facebook), location sharing networks (e.g., Foursquare). It will be very helpful for a tourist who is planning to hike Everest to “find the group of tourists who most frequently visit Everest.” Also, in social network, it will be very interesting to “find the social groups who are expert in suggesting music, movie or TV series.” Consider a social network where the vertices represent the users with relevant attributes and an edge between two vertices captures the social connection. The applications mentioned above can be represented by a query where a user needs to input some terms that captures her area of interest (e.g., “machine learning”, “Everest”, “music, movie or TV series”) and the result of such queries are communities consisting of experts in these specific areas. Also, a user can be

interested in finding more than one such community.

The queries mentioned above can be answered if we model the social network using an attributed graph as presented in Figure 1.1. Here, we present a small network of 20 authors (vertices) who work in the areas of “database” (DB), “machine learning” (ML), and “pattern recognition” (PR). Each author is associated with her fields of interest, and for each field of interest, she is assigned a score (a real number between 0 and 1) that represents how much influential the author is in this particular field. For example, author n_2 has very high influence in database (0.9). The edges in the network captures the interaction (i.e., co-authorship) among the authors.

Based on such attributed graphs, we present a novel parameter free influential community search query, namely Top- r Keyword-aware Influential Community Query (*KICQ*). To illustrate, let us consider the scenario where an aspirant Ph.D. student may be interested in finding the most influential community who are working in “ML” or “DB.” The *KICQ* returns the community C_3 as shown in Figure 1.1 as the most influential community (see Chapter 5 for the influential metric) since the members of the community have influence in either “ML” or “DB”, the community is dense and also contains highly influential members.

The studied problem can also help selecting appropriate customers for advertising. For example, a clothing company can target the top communities who are highly interested in “fashion”, and offer them some promotional package. Since, these communities can be considered as the trend setters in “fashion”, the clothing company can benefit from obtaining good review from these communities. Also, it can be applied for impromptu event organization where participants (having similar interests) will be selected based on the retrieved communities.

1.2 State of the Art and Research Gaps

Although community search has been a very popular research area for a long time, influential community search problem has received attention from the researchers recently, after Li et al. [7] introduced the notion of influence of a community. We present several significant research gaps in the previous works:

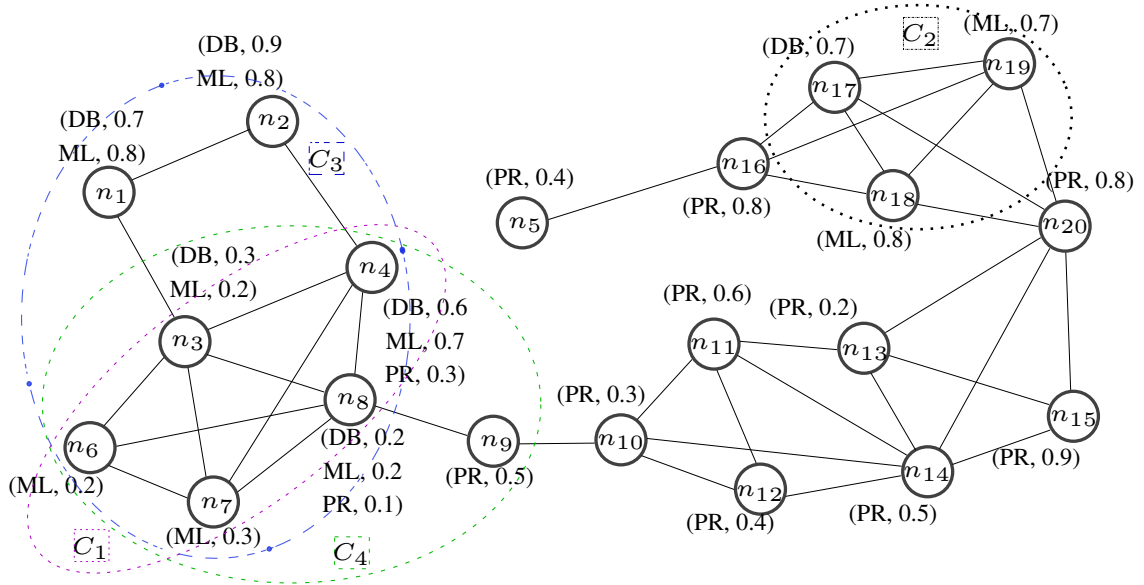


Figure 1.1: An attributed author-author graph, where each vertex has an associated list of attributes (keywords) and influences denoting her expertise. Different types of CS communities are marked as $C_1 - C_4$.

First, traditional CS works on an attributed graph require an input query vertex, and then find a group of neighboring vertices whose keywords have high similarity with the query vertex keywords. The resultant communities satisfy the required structural constraints [4, 6] (e.g., C_1 in Figure 1.1 with n_3 as the query vertex, and parameter $k = 4$ where k -truss is the structural constraint). A major limitation of such CS techniques is that the user needs to define the query vertex and the structural properties of the community explicitly, which might not be possible or suitable in many application domains. A couple of recent studies [9, 10] tried to address these limitations by finding cohesive (i.e., k -core or triangle density) communities having close similarity with query keywords. However, they do not consider the influence of individuals in different keywords (e.g., C_1 in Figure 1.1 is highly cohesive in terms of structure and keyword, but two highly influential vertices n_1, n_2 are ignored since influence is not considered) and also do not support flexible conjoining (using AND or OR predicates) of query keywords.

Second, existing works on influential community search only work on non-attributed graphs and also require specific values of structural parameters. For example, Li et al. [7] require users to mention the value of k while finding k -core based communities; similarly, Li et al. [8] require

the values of the minimum number of vertices (m) in a community and the maximum distance between any two vertices (p) while finding an mp -clique based community. Although such parameters allow high customization in search, we argue that the choice of these parameters highly depends on the internal structure of the graph in practice. For example, if k is set to a high value (e.g., 4) in a small graph (Figure 1.1), no community is returned by Li et al. [7] because there is no 4-core in this graph; and if k is low (e.g., 2), the community returned (e.g., C_2 in Figure 1.1) does not have high cohesiveness. Similarly, given a query vertex, Li et al. [8] only return the desired community under specific constraints of parameter values (e.g., C_4 in Figure 1.1 with parameters $m \leq 6$ and $p = 2$), which is impractical for an external user. Thus flexibility in fixing parameters while searching for desired communities is crucial.

Third, existing approaches to quantifying a community in terms of influence do not consider a comprehensive set of parameters that can affect the strength of a community. For example, the influence of a community is defined as the minimum influence among all members [7]; thus, a member with low influence can severely affect the influence of a community. We argue that an influence measure that considers both cohesiveness, the influence of individuals, and the size of the community should be considered while ranking communities, as all these factors contribute to the overall ranking of a community.

1.3 Challenges and Contributions

A major challenge in realizing the proposed *KICQ* query is in matching the expertise domains of vertices with the query keywords. Keywords are natural language text, and there can be many keywords with similar meaning. For example, two keywords “natural language processing”, and “computational linguistics” may look different, but these are very similar if we consider their semantics. So, designing a mechanism that can automatically understand the semantic similarity between keywords is necessary.

Another major challenge comes from the fact that communities and their influences need to be computed and compared on the fly based on the set of keywords in the query. The query can be any combination of multiple keywords, and it is impossible to pre-calculate scores

for all possible combinations, since there can be millions of keywords involved in a network. Thus, existing pre-computation based approaches are not suitable for our purpose. The idea of obtaining communities for a single keyword and later combining them is also computationally expensive. So, we need to design an influential community search algorithm that can efficiently handle multiple keywords during runtime.

Additionally, influence is a subjective measure. So, designing a measure that can capture the influence of a community is also a big challenge.

In this study, we address the above challenges, and the limitations in the state-of-the-art literature presented in Section 1.2. In summary, we make the following contributions:

First, we design *KICQ* in such a way that enables users to issue an influential community search query intuitively by merely using a set of query terms (words or phrases), and predicates (AND or OR) (addressing the first limitation). In this context, we propose a novel word-embedding based keyword similarity model that enables *semantic community search*, which substantially alleviates the limitations of *exact keyword* based community search. For example, a user may use “song” instead of “music,” where exact search fails to retrieve the relevant communities if the attributed graph does not contain “song” as a keyword. *This approach is of independent interest in enhancing any knowledge graph with semantically meaningful sets of words.* (Chapter 4)

Second, we propose a new influence measure for a community that considers both the cohesiveness and influence of the community and eliminates the need for specifying values of internal parameters of a network (addressing the second limitation). The influence measure also captures the influence of individual members in a better intuitive sense rather than the influence of the community being dominated by the minimum influence of a member (addressing the third limitation). We demonstrate the effectiveness of the proposed measure in a case study. (Chapter 5)

Third, we propose two efficient algorithms for searching influential communities in a large, attributed graph. The basis of the first algorithm is pruning the communities that cannot be a part of the answer set based on the computed scores of already explored subgraphs. The second one is a novel tree-based approach, where we augment the tree with influence score bounds for

each keyword and prune the unnecessary branches of the tree based on the scores of the explored community. (Chapter 6)

Fourth, we conduct comprehensive experiments with real datasets to evaluate our proposed algorithms. The experimental results show that our algorithms are highly efficient and effective in retrieving keyword aware influential communities compared to the state-of-the-art influential community search technique. (Chapter 7)

1.4 Organization

Now, we outline the organization of the rest of the book. First, we present the related works in Chapter 2. Then, we formulate the problem and present the overview of our proposed system in Chapter 3. In Chapter 4, we present the novel word-embedding based keyword similarity model. The influence measure for a community is presented in Chapter 5. Afterward, we describe the algorithms for influential community search in Chapter 6. We present extensive experiments on two real large graphs in Chapter 7, and discuss a case study in Chapter 8. Finally, we conclude the study in Chapter 9.

Chapter 2

Related Works

The notion of influential community search has been introduced by [7] in 2015 and since then, it has received significant interest. *Keyword search*, *cohesive subgraph mining*, and *community detection* are some well studied related problems. Several relevant but different works on *team formation in social networks* are also included in this chapter. Finally, we discuss several state-of-the-art studies on community search.

2.1 Keyword search

Keyword search is an extensively studied research topic, mostly for web searches. Zhang et al. [11] addressed the spatial keyword search problem using ubiquitous inverted index to retrieve objects of interest that are ranked based on both their spatial proximity to the query location as well as the textual relevance of the object's keywords. Li et al. [12] proposed and defined the problem of keyword based correlated network computation over a massive graph. Bhalotia et al. [13] proposed the first backward search algorithm. They defined BANKS, a system which enables keyword-based search on relational databases together with data and schema browsing. Kacholia et al. [14] improved this approach by proposing a bidirectional search algorithm. He et al. [15] proposed BLINKS, a bi-level indexing scheme to find top- k keyword search on graphs. They divided the graph into blocks and used both intra-block indexing and inter-block indexing to make the search procedure faster. However, these works do not address the problem of finding

communities.

2.2 Cohesive subgraph mining

Cohesive subgraph mining involves extracting dense subcomponents from graphs. Cohesive subgraphs like *maximal cliques* [16], *k-core* [17], *k-truss* [18], etc. are widely used techniques for social network analysis. Moody and White [19] worked on structural cohesion in social networks, also known as *maximal k-edge connected subgraphs*. These cohesive subgraphs form the basis of modeling communities.

2.3 Community detection

The task of finding communities can be divided into two major classes: community detection (CD), and community search (CS). In CS, communities are defined based on the query, and CS solutions aim to find communities efficiently in an online manner. CD methods usually use global criteria to detect all the communities from an entire graph, where the focus is more on quality (e.g., cohesiveness) than efficiency. Link based analysis was popular in initial studies [20] that did not consider attributes in a graph. Clustering based techniques [21–24], and topic modeling [25,26] are used in recent studies on attributed graphs. Studies on overlapping community detection [27,28] address the fact that a person can be a part of several communities (e.g., family, friends, co-workers). However, none of the studies enables a user to find specific communities of her interest, which is the main focus of our study.

2.4 Team formation in social networks

Team formation is the task of finding a subset of available individuals to complete a project which requires a specific set of skills. There has been significant works in this area but most of them do not consider the presence of social network of individuals [29]. Lappas et al. [30] introduced the inclusion of social networks. Li et al. [31] generalized the problem where there

CS Approaches	Simple graph	Attributed graph	
		Keyword	Others
Basic CS	[1, 35–40]	[4, 6, 9, 10, 44, 45]	[5, 46–48]
Influential CS	[7, 8, 41–43]	-	[49]

Table 2.1: Existing works on community search

is a requirement of a specific number of experts for each task. Aris et al. [32] studied the case where tasks arrive in an online manner and the workload is balanced among people. Kargar and An [33] studied another variant which finds top- k teams. A slightly different but relevant problem is selecting a group of individuals for an impromptu activity based on their location and social interaction studied by Yang et al. [34]. Team formation can be viewed as a set coverage problem, usually with a minimum communication cost objective. These works differ from community search, as a team does not need to be cohesive (members are densely connected to each other).

2.5 Community search

We present different directions of CS studies in Table 2.1. Most of the basic CS studies on simple graphs [1, 35–39] find communities containing given query vertices. Li et al. [40] studied persistent communities in a temporal network, in which every edge is associated with a timestamp. Li et al. [7] introduced the notion of influential CS where vertices are assigned an influence score, and the influence of a community is modeled as the minimum influence of the members. Chen et al. [41] and Bi et al. [42] developed faster algorithms to solve the same problem. Zheng et al. [43] studied influential CS in an undirected weighted graph, where the weight of an edge represents the semantic intimacy between two vertices. Li et al. [8] defined a community in terms of kr -clique and designed algorithms to retrieve the most influential community. All of these studies ignore rich information of vertices found in attributed graphs and require several vertices or internal parameters as part of a query, which is very difficult for a user who does not have enough knowledge of the graph.

There are several studies on CS in attributed graphs. Fang et al. [4] proposed the ACQ

algorithm to find subgraphs satisfying structural and keyword cohesiveness. Huang et al. [6] also explored attribute driven CS in terms of k -truss. Chen et al. [46] studied CS in an attributed graph where each vertex has a *profile*: a set of keywords arranged in a tree structure. Chobe et al. [44] employed keyword search techniques to facilitate CS in attributed graphs. However, these studies also require a set of vertices and/or internal parameters as part of the query. Few recent works [9, 9, 10] study keyword-based CS that take a set of keywords as input and return a subgraph as the community that has the *best* match with the given set of query keywords. In these works, the cohesiveness of the subgraph is measured differently, i.e., k -core in Zhang et al. [9], triangle density in Chen et al. [10], and average proximity in Khan et al. [45]. To decide a single best-matched subgraph, they define functions that consider the presence or absence of keywords and structural cohesiveness in the subgraph. These studies are different from ours as they only consider the presence or absence of keywords in different vertices of the subgraph and cannot be adapted for the scenario where we need to rank the communities and each vertex has a certain degree of influence in each keyword. There are CS studies on spatial graphs [5, 47, 48] as well, which are of different interest to our problem.

Li et al. [49] studies skyline community search where each vertex is associated with a d -dimensional influence score. However, their study is designed for low values of d (i.e., $d < 5$). With $d = 5$, their algorithms require more than 10^3 seconds in a graph with half a million vertices. This study cannot be extended for an attributed graph where vertices are associated with influence scores in multiple keywords, because there can be millions of keywords (dimension) in such an attributed graph. Also, this approach aims to find communities with a global objective function, and there is no way to search communities of specific interest.

To the best of our knowledge, there is no CS study addressing both keywords and influence scores simultaneously. Also, we introduce CS with semantic keywords, which relieves the users from the burden of putting the keywords exactly as in the attributed graphs.

Chapter 3

Problem Definition and System Overview

In this chapter, we first define the attributed graph, propose our community model, and define keyword aware influential community query (*KICQ*), a novel query to find the most influential communities from the attributed graph. After that, we present a high level overview of the proposed system.

3.1 Problem definition

Definition 1 (*Attributed graph*) An attributed graph $G^+(V, E, A)$ is an undirected graph, where V is the set of vertices, E is the set of edges, and each vertex v is associated with a set of attributes A_v .

Definition 2 (*Influential community*) Influence means the power of affecting someone's emotion, activity or thinking. An influential community is a group of individuals who have close connection among them, and can influence other individuals in the network.

Influential community consists of a group of closely connected individuals whose activities are followed by a significant number of individuals in the network. For example, a group of researchers who are dominating in some topic (keyword) act as the trend maker in that topic. Any new researcher of that topic might follow their works constantly and his/her activities are significantly influenced by that group. In this study, we will consider the influence of an

individual person on a topic as a numerical value between 0 and 1. The more the value is, the higher the influence of the person is in that topic.

We use an attributed graph to model a social network where the vertices V represent the set of individuals in the network, and each edge $e \in E$ captures the social connection between two individuals. Each vertex v is associated with a set of tuples of the form $A_v = \{(w_i, s_v(w_i))\}$, where w_i is a keyword and $s_v(w_i) \in [0, 1]$ is the influence score of vertex v in keyword w_i .

Example 1 A co-authorship network can be formed from a corpus of research articles and the set of keywords can be identified from the frequently tagged keywords in each article. Each author is associated with the keywords relevant to her fields of studies, and for each keyword, the number of publications are considered while calculating her influence score. Let us consider the attributed graph presented in Figure 1.1. Here the set of keywords are $\{“DB” (Database), “ML” (MachineLearning), “PR” (PatternRecognition)\}$. Now, n_1 is an author who published 35 research articles in “social network analysis” for which relevant keywords are “DB”, and “ML”. n_1 has also published 5 articles in “artificial neural network” and “ML” is the only relevant keyword. So, n_1 has total 40 publications relevant to keyword “ML” and 35 publications relevant to “DB”. Therefore, n_1 has higher influence score in “ML” (e.g., 0.8) than in “DB” (e.g., 0.7).

We consider the connected components of maximal k -cores as the *influential communities*, where the *influence* is defined as Equation 5.2 (see Section 5). k is termed as *cohesion factor* throughout the article.

Definition 3 (maximal k -core) Let H be a subgraph of G^+ , induced by the set of vertices $V_H \subseteq V$. Let the degree of a vertex v in H is denoted by $deg_H(v)$. H is a k -core if $\forall_{v \in V_H} deg_H(v) \geq k$, where k is a non-negative integer. H is a maximal k -core if there is no super k -core in G^+ that contains H .

Example 2 In Figure 1.1, each of the vertices in the subgraph induced by $\{n_3, n_4, n_6, n_7, n_8\}$ has degree at least 3. Thereby, they form a 3-core. Also, there is no super 3-core that includes all these vertices. So, these vertices form a maximal 3-core. As they are connected, this subgraph can be considered as a candidate influential community.

Now, we define the keyword-aware influential community query, *KICQ* as follows.

Definition 4 (*KICQ*) Let $G^+(V, E, A)$ be an attributed graph, and $q(T, P)$ be a query tuple where $T = \{t_1, t_2, \dots, t_n\}$ is a set of terms (i.e., words or phrases) and P is a predicate (AND, OR) for conjoining the query terms. Let k_{min} be the minimum cohesion factor for being a candidate community, and r be a positive integer specifying the number of top communities to be returned. Then we form the *KICQ* as a quadruple (X, P, r, k_{min}) , where $X = \{X_1, X_2, \dots, X_n\}$ and X_i is the set of semantically similar keywords (see Chapter 4) of term t_i . *KICQ* finds r most influential communities H_1, H_2, \dots, H_r from G^+ .

3.2 System overview

An overview of our system is presented in Figure 3.1. The system is mainly divided into two phases. First, we construct a keyword-aware attributed graph from a social network corpus that may consist of structured and/or unstructured (i.e., text) data. In an academic domain, the corpus can be scientific publications of researchers (e.g., authors, titles, abstracts, author-provided keywords, etc). Second, we focus on searching keyword-aware influential communities using the constructed attributed graph, given a query as a set of terms and predicates.

The distinctive features of the first phase are as follows:

STEP 1 : First, we build an attributed graph from a domain corpus by extracting entities of possible vertices and edges to represent the social network. We also extract the keywords by finding the most frequent tags/terms in the corpus. One example of such a graph can be seen as an academic graph, where vertices represent authors and an edge represent co-authorship between two authors. Each node is associated with the relevant keywords (i.e. attributes) of the corresponding author. Here, such keywords represent the expertise topics of the authors in the graph. Each node is further given a score for each keyword that represents its degree of expertise.

STEP 2 : To enable keyword-aware influential community search, we augment keywords with their semantically related terms and keywords. These augmented vocabulary will be used

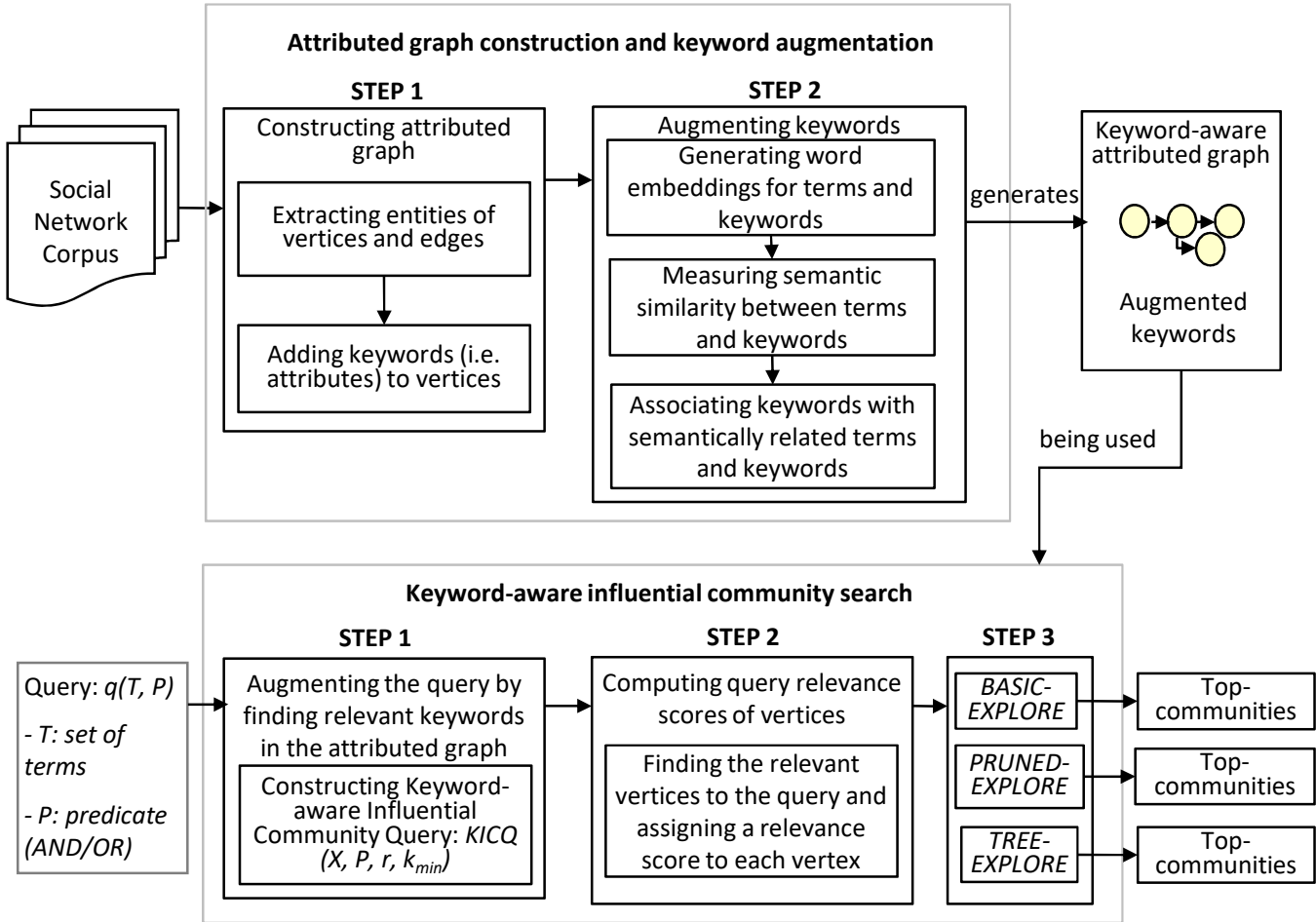


Figure 3.1: The overview of the proposed system

for identifying X in $KICQ$ (Definition 4). For this purpose, we build *word embedding vectors* as an external knowledge source for associating keywords in the graph with semantically related terms and keywords. These vectors are generated from a large text corpus relevant to the source of the graph using the algorithm [50] that has shown the power of encoding semantics of words. The semantic relatedness is estimated by exploiting the word embedding vectors. These augmented keywords will enhance the capability of our community search algorithms. The output of this step is a graph, called *keyword-aware attributed graph*.

Further, the unique features of the second phase of our system can be briefly highlighted below:

STEP 1 : Initially, a query raised by a user is given in the form of a pair $q(T, P)$ consisting of a set of query terms T , and a predicate P . The terms need to match with the keywords in the attributed graph to find meaningful communities. We acknowledge the difficulty faced by the users to put the exact terms while raising a query. For example, it is highly likely that some of the users will input “song” instead of “music”. To help the users to easily raise a query, we augment each query term with a semantically meaningful set of keywords. The output of this step is a *KICQ*.

STEP 2 : Given a *KICQ* query, our objective is to find the vertices relevant to the query and then compute their query relevance scores (Equation 5.1). Relevance score of vertices are used to compute the scores of potential influential communities. We argue that a measure that rewards both the cohesiveness of the community and high influence of the members, and does not require user input of any internal parameters (e.g., k in k -core) is more preferable than the existing influence measures. We propose a linear weighted summation of the cohesiveness of the community and the total influence of the members of the community to estimate the overall score of a community (Section 5).

STEP 3 : Given the augmented query and the influential score function, our focus is now to retrieve top- r most influential communities relevant to the query. Since we are the first to propose the keyword-aware influential community search problem, and existing pre-computation based approaches are not suitable to retrieve communities for any given query, we first present a basic solution named *BASIC-EXPLORE* followed by two efficient algorithms: *PRUNED-EXPLORE* and *TREE-EXPLORE*.

Chapter 4

Constructing Attributed Graph

In a social network, each individual has expertise in various fields that can be represented by terms (any combination of words in natural language). There are millions of such terms in a large network and it is unlikely for users to come out with the exact terms while raising a query. We design a semantic keyword similarity model which can capture the semantic meaning of a term by converting it into a word vector. Then several measures can be used to compute the similarity among these word vectors, thereby allowing us to find semantically relevant terms to any given term.

In this chapter, we first present the semantic keyword similarity model, and then discuss how we can generate the attributed graph and augment the query terms with keywords using this model.

4.1 Semantic keyword similarity model

In this section, we present how to augment keywords that can be used to form a *KICQ*. Given a term, finding its semantically related keywords is not trivial; basic preprocessing like removing whitespaces and stopwords can be helpful but there are still some major concerns. Two or multiple terms with syntactical difference can indicate a similar or the same keyword (e.g., “error detection and error correction” and “error detection and correction”). Even two different terms can represent the same keyword (e.g., “AI” and “artificial intelligence”). Also,

some terms are semantically very related to each other (e.g., “neural network” and “gradient descent optimization”).

We first train a word embedding model to generate an embedding vector of any given term (and also keyword). Then, we use such a vector to capture the semantics of a term and find its relevant keywords. For training this model, we collect a domain corpus (e.g. scientific publications in an academic domain), where each document is already associated with keywords. We call this *training corpus*. Among the keywords in the training corpus, we select N keywords with the highest document frequencies (a document frequency is defined as the number of times a keyword appears in all the documents in the corpus). Then, our focus is to develop a model that can find the relevant keywords out of these N -top keywords given a term by using the word embedding model.

Word embedding is a learning technique that can enable words or phrases to map to vectors of real number. Word2vec [51] is a representative word embedding model that can be trained to construct linguistic contexts of words and thereby generate such a mapping. The output of Word2vec is a vector space where similar words are positioned close to one another. It has been well demonstrated that Word2vec has many advantages over earlier traditional embedding techniques for semantic analysis of words [51].

To build a Word2vec model on the training corpus, we perform tokenization from each text document on the training corpus. This process includes removing stopwords, and extracting only nouns, adjectives and gerunds as usually important concepts are represented via nouns [52]. Also, some keywords consist of adjectives and gerunds (i.e., **expert** systems, machine **learning**, etc.). After tokenization, we convert each of the extracted words into lemmatized form to identify its single canonical form. Then, we feed the extracted words into a Word2Vec model that represents each word as a vector. Using similarities between two vectors, we can estimate how close and distant they are in terms of semantic meaning.

Now, we represent a keyword or term as a vector. A keyword/term can be thought of as a phrase containing one or many words. In our approach, the representative vector is formed using the average of the embedding vectors of the constituent words. By doing so, we are now ready to estimate the similarity between any term and any keyword according to their vectors.

Given any two terms (or two keywords, or a term and a keyword) t_1 and t_2 , we denote their embedding vectors as x_{t_1} and x_{t_2} , respectively. In the following, we present three approaches to estimating a similarity between these two terms, denoted as $S(t_1, t_2)$:

- **cosine**

The cosine similarity of two vectors \mathbf{x} and \mathbf{y} is defined as, $\text{cosine}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}^T \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}$, where \mathbf{x}^T denotes the transpose of vector \mathbf{x} and $\|\mathbf{x}\|$ denotes the $L2$ norm. The similarity of terms t_1 and t_2 is calculated by the cosine similarity of their embedding vectors x_{t_1}, x_{t_2} .

$$S(t_1, t_2) = \text{cosine}(x_{t_1}, x_{t_2}) \quad (4.1)$$

- **normalized maximum**

Given a term t , its word vector V^t is denoted as $V^t = [(w_1^t, s_1^t), (w_2^t, s_2^t), \dots, (w_L^t, s_L^t)]$, where w_i^t is the i^{th} most similar word to x_t , s_i^t is the corresponding similarity score, and L is the number of similar words considered. Now, given two terms t_1, t_2 and their word vectors V^{t_1}, V^{t_2} , we use the following formula proposed by Kang et. al. [53] to calculate their similarity. This measure has been used to measure a similarity between two sentences, where each sentence consist of multiple words. In a sense, a sentence is a word vector in our context.

$$S(t_1, t_2) = \frac{\sum_{(w_1, s_1) \in V^{t_1}} \text{sim}_{max}(w_1, V^{t_2}) + \sum_{(w_2, s_2) \in V^{t_2}} \text{sim}_{max}(w_2, V^{t_1})}{|V^{t_1}| + |V^{t_2}|} \quad (4.2)$$

Here, $\text{sim}_{max}(w, V^t) = \max_{(w_i, s_i) \in V^t} \text{sim}(w, w_i)$ and $\text{sim}(w, w_i)$ can be calculated by measuring the cosine similarity of the embedding vectors of w and w_i .

- **indirect cosine**

Given two terms t_1, t_2 and their word vectors V^{t_1}, V^{t_2} , we first construct a vocabulary, U

combining words from both vectors. Formally,

$$U = \{w : (w, s) \in V^{t_1} \text{ or } (w, s) \in V^{t_2}\} \quad (4.3)$$

To simplify our notation, let U be denoted as $U = \{w_1, w_2, \dots, w_n\}$, where $n = |U|$. Now, we define another vector $SV^t = [s_1, s_2, \dots, s_n]$, where s_i is the similarity score of term t to word $w_i \in U$, which can be found from V^t . If $(w_i, s_i) \notin V^t$, s_i is set to 0. Finally, $S(t_1, t_2)$ is calculated as,

$$S(t_1, t_2) = \text{cosine}(SV^{t_1}, SV^{t_2}) \quad (4.4)$$

In our evaluation in Section 7.2, we compare the above three methods and choose the best one to augment the attributed graph. Finally, for any term t , we calculate its similarity with all the keywords in the given attributed graph, and find M -top most relevant keywords X_t ranked based on the similarity scores. M is a system configurable parameter. By default, M is set to 10.

4.2 Constructing attributed graph

In attributed graph G^+ , a vertex v is associated with a set of keywords. For all vertices, we extend each keyword t with its M -top most relevant keywords X_t using the semantic similarity model. For any keyword $w \in X_t$, the influence score of vertex v , $s_v(w) = s_v(t)$ since w and t are semantically similar.

4.3 Augmenting keywords for KICQ

A query $q(T, P)$ consists of a set of terms $T = \{t_1, t_2, \dots, t_n\}$ and a predicate P . First, the semantic similarity model is used to augment each term t_i with the set of relevant keywords X_{t_i} . Then the system parameters r and k_{min} are used to formulate the keyword aware influential community query, $KICQ(X, P, r, k_{min})$ where $X = \{X_{t_1}, X_{t_2}, \dots, X_{t_n}\}$.

Chapter 5

Influential Community Measures

Given a keyword-aware influential community query, $KICQ(X, P, r, k_{min})$, we need to find top- r communities based on their influence measure. To define such measures, we first define the relevance of a vertex to a $KICQ$ query. Then, we present the desired properties of such communities and finally, propose a scoring function that can capture the influence of a community.

5.1 Query relevance score of a vertex

First, for a given query, we redefine the influence of a vertex based on its relevance to the query. The query relevance score γ_v of a vertex v is estimated as follows: each vertex v in the attributed graph is annotated with keywords and their influence score for the corresponding keywords, i.e., $(w_i, s_v(w_i))$. To estimate the relevance score, $\gamma_v \in [0, 1]$, we need to consider the list of semantic keywords X and the predicate P in the $KICQ$ query. Formally, we use the following definition for computing γ_v :

$$\gamma_v = \mathbf{f}_{X_{t_i} \in X}[\mathbf{g}_{w \in X_{t_i}} s_v(w)] \quad (5.1)$$

Here, \mathbf{f} and \mathbf{g} are two aggregate functions: \mathbf{g} combines the relevance of the vertex for the semantic keywords of a query term, and \mathbf{f} combines the relevance scores in all terms considering

the predicate P . We use \mathbf{g} as the aggregate function, MAX ; whereas we use \mathbf{f} as MIN for AND predicate and MAX for OR predicate, respectively. MIN aggregate ensures that a vertex has high relevance to all the terms, while MAX only requires high relevance to any of the terms.

Example 3 Consider a query $q(\{\text{"social network analysis"}, \text{"graph mining"}\}, AND)$. The query can be interpreted as **find top- r influential communities with members who have expertise in "social network analysis" AND "graph mining"**. Now, the terms mentioned in the query does not exactly match the keywords(i.e., "DB", "ML", and "PR") in the attributed graph in Figure 1.1. We first find the relevant keywords for the terms in the query. Say, the relevant keywords are $\{\text{"DB"}, \text{"ML"}\}$ for "social network analysis", and $\{\text{"DB"}, \text{"PR"}\}$ for "graph mining". Let, $r = 3$ and $k_{min} = 2$. So, the augmented query becomes $KICQ(\{\{\text{"DB"}, \text{"ML"}\}, \{\text{"DB"}, \text{"PR"}\}\}, AND, 3, 2)$. The interpretation is, we need to find top **3** (r) most influential communities with members who have some influence in keywords "**DB**" or "**ML**" (for term "social network analysis") **AND**(predicate P) in keywords "**DB**" or "**PR**" (for term "graph mining"). In such communities, each member must be connected to at least **2** other members (k_{min}).

Let us consider the aggregate functions \mathbf{g} be maximum and \mathbf{f} be minimum. Then the relevance of n_1 to term "social network analysis" will be the maximum among 0.7 (for DB) and 0.8 (for ML), which is **0.8**. Similarly, the relevance of n_1 to term "graph mining" becomes maximum of 0.7 (DB) and 0.0 (PR), i.e., **0.7**. Finally, the relevance of n_1 to the query is minimum of 0.8 and 0.7, which is **0.7**.

5.2 Desiderata of an influential community

The influence measure of a community is mostly subjective. However, the following properties are desired from an influential community.

(1) **Connectivity.** A good community must be connected.

(2) **Cohesiveness.** Cohesion of a community is defined as the number of nodes that needs to be removed to disconnect it. A community is desired to have high cohesion and therefore high density.

(3) **Highly influential individuals.** A good community should consist of individuals who are highly influential in their field of expertise.

(4) **Large.** large communities are preferred, if the connectivity and high cohesion of a community can be retained.

5.3 Influential score function

We use a linear weighted summation of the cohesiveness and influences to calculate the overall score of a community. Let $H = (V_H, E_H)$ is a subgraph of attributed graph $G^+(V, E, A)$. If H is a community (connected component of maximal k -core), then the score of H is:

$$\zeta(H) = \beta \times \underbrace{\frac{k}{\max\text{-deg}(G^+)}}_{\text{cohesiveness score}} + (1 - \beta) \times \underbrace{\frac{\sum_{v \in V_H} \gamma_v}{|V|}}_{\text{influence score}} \quad (5.2)$$

Here, $\max\text{-deg}(G^+)$ is the maximum degree of all vertices in G^+ . Both the cohesiveness and the influence score of a community are normalized within $[0, 1]$, and the preference parameter $\beta \in [0, 1]$ defines the importance of one score relative to the other. A large value of β leads to retrieval of the communities with more cohesiveness while a small value promotes communities with highly influential individuals. β is a system configurable parameter. Suppose, the user sets a ratio of the weight of *cohesiveness score* to the weight of *influence score* as 1:2. So, $\frac{\beta}{1-\beta} = \frac{1}{2}$, i.e., $\beta = \frac{1}{3}$.

Finally, we discuss the benefits of choosing such a scoring function. Since we model a community using connected k -core, connectivity and cohesiveness is ensured. $\max\text{-deg}(G^+)$ and $|V|$ is constant for attributed graph G^+ . Thus the influence score of community H depends on $\sum_{v \in V_H} \gamma_v$ which prefers large community with highly influential individuals. Also as long as some low influential members do not disrupt the cohesiveness of the community, the score of this community is not penalized, which is the case in Li et al. [7]. We acknowledge that such a measure is not unique, and other measures can be explored in the future. However, experiments using real datasets and the case study presented in Chapter 8 demonstrate that our proposed influence measure can capture cohesive communities with highly influential members.

Chapter 6

Algorithms for Influential Community

Search

In this chapter, we present algorithms for finding r most influential communities from the attributed graph G^+ for a given $KICQ(X, P, r, k_{min})$. Since the notion of *influential community* changes with different sets of query keywords, existing pre-computation based approach [7, 8] cannot be adapted for this purpose.

6.1 A straightforward approach, BASIC-EXPLORE

A straightforward approach to answer $KICQ$ on a large graph is as follows. First, we extract the subgraph, which we call the *query essential subgraph*, G_q , containing vertices and edges that are relevant to the query. Then we find all the connected components of maximal k -core subgraphs for all possible values of k . Finally, we return the top r communities having the highest influential community scores (as per Equation 5.2).

6.1.1 Finding query essential subgraph

The query essential subgraph, $G_q(V_q, E_q, \gamma)$ is a subgraph of the attributed graph $G^+(V, E, A)$ induced by V_q , the set of vertices with a non-zero query relevance score. In G_q , each vertex v is

Algorithm 1 BASIC-EXPLORE (G_q)

```

1: compute core decomposition for all vertices in  $G_q$ 
2: initialize a priority queue  $Q$  with  $r$  empty communities (score 0)
3: for  $k = k_{min}$  to  $max-deg(G_q)$  do
4:    $H^k =$  maximal  $k$ -core in  $G_q$ 
5:    $CC^k =$  set of connected components in  $H^k$ 
6:   for all  $h(V_h, E_h) \in CC^k$  do
7:      $\zeta(h) =$  score of  $h$ 
8:     if  $\zeta(h) >$   $r^{th}$  best score then
9:        $Q.pop()$ 
10:       $Q.push(h)$ 

```

annotated with its relevance score γ_v , and E_q is the set of edges between any two vertices in V_q . To efficiently generate the G_q , we maintain an inverted index, where for each keyword w , a list IL_w of the vertices that contain w is stored. Thus, for a given $KICQ$ query, V_q can be obtained by,

$$V_q = \begin{cases} \bigcap_{X_{t_i} \in X} [\bigcup_{w \in X_{t_i}} IL_w], & \text{if } P = AND \\ \bigcup_{X_{t_i} \in X} [\bigcup_{w \in X_{t_i}} IL_w], & \text{otherwise} \end{cases} \quad (6.1)$$

After retrieving V_q , we compute the query relevance score of each vertex $v \in V_q$ (Equation 5.1) and retrieve E_q that denotes the connections between all pairs of vertices in V_q .

6.1.2 Finding k -cores and most influential communities

Algorithm 1 outlines the procedure BASIC-EXPLORE for finding r most influential communities from G_q . First, we compute core decomposition for all vertices in G_q using the $O(|E_q|)$ algorithm proposed by Batagelj et al. [54]. A priority queue Q is used to hold our solution. We initialize Q with r empty communities having 0 score. In our case, a community must be at least k_{min} -core. Again, the maximum cohesion factor of a community in G_q can be $max-deg(G_q)$, since there is no vertex in G_q with a higher degree. Thus we need to first find all connected components of maximal k -cores from G_q , where the value of k is in range $[k_{min}, max-deg(G_q)]$. Then, we compute the influential scores of each computed community, and finally, maintain the top- r communities in Q ordered by the scores of the communities.

Example 4 To demonstrate how the basic exploration algorithm works, consider the query

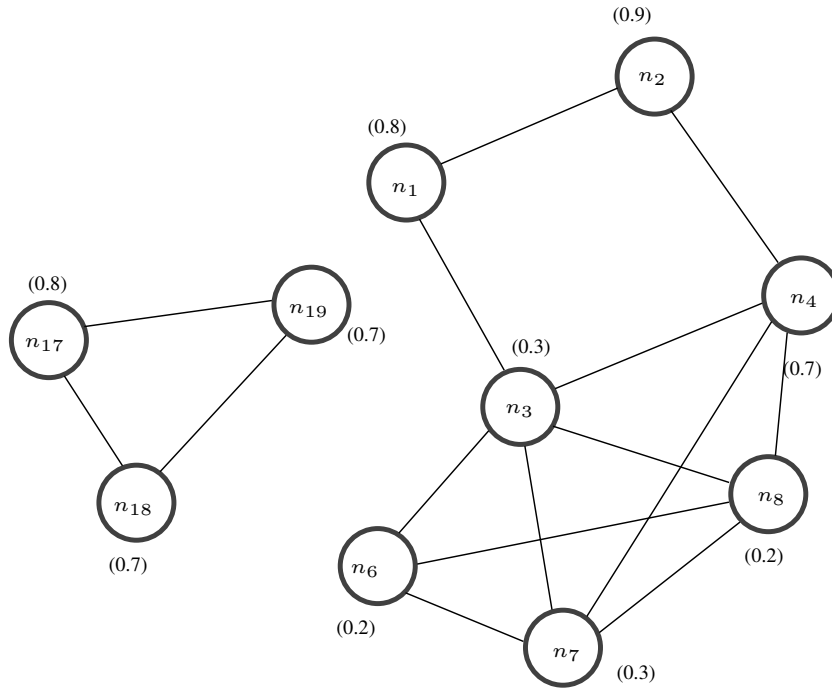


Figure 6.1: A query essential subgraph.

“social network analysis”, for which the KICQ is $(X = \{“DB”, “ML”\}, P = NONE, r = 3, k_{min} = 2)$. We first identify the query essential subgraph G_q (showed in Figure 6.1) given the attributed graph as in Figure 1.1. Here $|V| = 20$. Since $k_{min} = 2$ and $max-deg(G^+) = 6$, the value of k can be any integer between 2 and 6. The connected components of 2-core are $h_1 = \{n_1, n_2, n_3, n_4, n_6, n_7, n_8\}$ and $h_2 = \{n_{17}, n_{18}, n_{19}\}$. The cohesiveness score is $2/6 = 0.333$ for both of the communities. The influence score of h_1 is $(0.8+0.9+0.3+0.7+0.2+0.3+0.2)/20 = 0.17$ and h_2 is 0.11. If β is set to $1/3$ (for example), then the score of the communities are 0.224 and 0.1844 respectively. Again, $h_3 = \{n_3, n_4, n_6, n_7, n_8\}$ is the only connected component of 3-core with score 0.2233. The procedure BASIC-EXPLORE terminates as there is no 4-core in the G_q . So, according to the proposed scoring function, h_1 is the most influential community while h_3 and h_2 rank second and third respectively.

6.1.3 Time complexity

Finding the relevant vertices and calculating their relevance score can be done in $O(|V_q| \times N_w)$ time, where $N_w = \sum_{X_i \in X} |X_i|$ is the total number of relevant keywords. If the graph

is implemented with adjacency list, E_q can be obtained in $O(|V_q|)$ time by taking union of adjacency list and V_q for each vertex. So, time complexity for computing G_q is $O(|V_q| \times N_w)$.

Core decomposition of V_q is done in $O(|E_q|)$ time. The operations (push, pop) performed in the priority queue takes $O(\log r)$ time. So, the time required for initializing Q is $O(r \log r)$. Exploring a maximal k -core requires computing its connected components ($O(|V_q| + |E_q|)$), obtaining k -core vertices ($O(|V_q|)$), computing scores of each connected components, and updating the priority queue Q . For any community $h(V_h, E_h)$, the run-time for computing its score is bounded by $O(|V_h|) = O(|V_q|)$ (simplified). So, if N_k is the number of influential communities with cohesion factor k , then the runtime of exploring all k -cores is bounded by $O(\max\text{-deg}(G_q) \times ((|V_q| + |E_q|) + N_k \times (|V_q| + \log(r))))$.

Considering $|V_q| > \log(r)$, the bound can be simplified as $O(\max\text{-deg}(G_q) \times |V_q|^2)$ for a dense graph¹. Since, this dominates the time complexity of finding G_q , we can conclude that, the overall complexity of BASIC-EXPLORE is $O(\max\text{-deg}(G_q) \times |V_q|^2)$.

6.2 Pruned exploration approach, PRUNED-EXPLORE

The major bottleneck of BASIC-EXPLORE is that it needs to explore all maximal k -cores, for different values of k , and find the connected components of each maximal k -core subgraph. Such exploration is computationally expensive for a large graph. Instead of directly exploring the subgraphs to compute the maximal k -core and its connected components (communities), we first estimate the upper bound score of the communities of the corresponding subgraph. This bound can be used to prune a large number of redundant subgraphs that cannot be a part of the top- r influential communities.

First, we find the query essential subgraph G_q , compute core decomposition, and initialize priority queue Q as described in Section 6.1. Now, we need to explore G_q to retrieve communities for all possible values of k . As discussed before, the value of k must be between k_{\min} and $\max\text{-deg}(G_q)$. We propose the following lemmas, which pave the foundation of our pruning.

¹ $N_k < |V_q|$ and for any dense graph $G(V, E)$, $|E|$ is $O(|V|^2)$

Algorithm 2 PRUNED-EXPLORE (H, k)

Input: Subgraph of G_q $H = (V_H, E_H)$, cohesion factor k .

```

1:  $min-deg(H) = \min_{v \in V_H} (deg_H(v))$ 
2: if  $min-deg(H) > k$  then
3:    $k = min-deg(H)$ 
4:  $H^k =$  maximal  $k$ -core in  $H$ 
5:  $CC^k =$  set of connected components of  $H^k$ 
6: for all  $h \in CC^k$  do
7:   if actual score,  $\zeta(h) > r^{th}$  best score then
8:      $Q.pop()$ 
9:      $Q.push(h)$ 
10:  for  $k' = k + 1$  to  $max-deg(G_q)$  do
11:    if upper bound score,  $\zeta_{k'}^*(h) > r^{th}$  best score then
12:      PRUNED-EXPLORE( $h, k'$ )
13:    break

```

Lemma 1 Let, $H(V_H, E_H)$ be a subgraph of G_q . For any community in H , the maximum influence score can be the sum of the query relevance scores of all vertices in H . Thus, without computing the vertices of k -core subgraph, we can calculate the upper bound of the score of any community in H for a particular value of k as follows.

$$\zeta_k^*(H) = \beta \times \frac{k}{max-deg(G^+)} + (1 - \beta) \times \frac{\sum_{v \in V_H} \gamma_v}{|V|} \quad (6.2)$$

Lemma 2 If $H = (V_H, E_H)$ is a subgraph and $min-deg(H) = \min_{v \in V_H} (deg_H(v)) > k$, then any community in H must be at least $min-deg(H)$ -core.

According to Lemma 1, we can prune a subgraph if its upper bound score is lower than the r^{th} best score of already retrieved communities from G_q . Moreover, Lemma 2 helps us to avoid the computation of certain cores from G_q .

Now, we develop a recursive procedure PRUNED-EXPLORE to search for influential communities in G_q . Algorithm 2 outlines the procedure. Initially, PRUNED-EXPLORE(G_q, k_{min}) is called to extract communities with minimum cohesion factor. In later steps, the procedure is recursively called to extract communities with higher cohesion factors.

Let us consider that we want to find communities with cohesion factor k , from a subgraph $H(V_H, E_H)$ of G_q . In lines 1-3, we determine the minimum degree of the vertices in H , $min-deg(H)$. If $min-deg(H) > k$, we set $k = min-deg(H)$ and directly compute such k -cores (according to Lemma 2). In lines 4-5, we find the set of connected components of maximal k core of

H , denoted by CC^k . The loop in line 6 runs for each connected component. We update the priority queue if any connected component's score is higher than the current top- r communities in lines 7-9. We explore the connected component for higher values of k in lines 10-13. We use Lemma 1 to prune exploration for the values of k for which the upper bound of the score is lower than the r^{th} best community. When the procedure terminates, the queue holds the final top- r communities.

Example 5 Suppose, we want to explore the query essential subgraph G_q in Figure 6.1 and our goal is to find top-2 communities ($r = 2$). Initially, queue contains two empty communities with score 0, i.e., r^{th} best score = 0. In the original attributed graph G^+ (Figure 1.1), the number of vertices in the original attributed graph (Figure 1.1), $|V| = 20$ and maximum degree of vertices, $\max\text{-deg}(G^+) = 6$. Let's assume the value of system parameter β is $\frac{1}{3}$. Initially $\text{PRUNED-EXPLORE}(H = G_q, k = k_{\min} = 1)$ is called.

(i) Here, $\min\text{-deg}(QEG) = 2 > k$. So, we set $k = 2$ and start extracting communities which are connected components of a 2-core. We find the connected components of maximal 2-core $CC^2 = \{h_1 = \{n_1, n_2, n_3, n_4, n_6, n_7, n_8\}, h_2 = \{n_{17}, n_{18}, n_{19}\}\}$ with scores 0.224 and 0.1844 respectively (Eqn. 5.2). We update the queue accordingly and now, r^{th} best score = 0.1844. The upper bound score of a subgraph must be higher than 0.1844 to be considered for exploration.

(ii) Then, we explore h_1 for communities with cohesion factor $k = 3$ as the upper bound 0.28 is greater than the current 2nd highest score (0.1844). After computing the maximal core and its connected components, we find a community $h_3 = \{n_3, n_4, n_6, n_7, n_8\}$ with score 0.2233. The queue is updated and the current 2nd highest score is 0.2233.

(iii) Now $\zeta_4^*(h_3) = 0.4467$ indicates that there can be potential top communities in h_3 for $k = 4$, but the procedure $\text{PRUNED-EXPLORE}(h_3, 4)$ terminates as there is no 4-core found from this subgraph.

(iv) Similarly, $\text{PRUNE-AND-EXPLORE}(h_2, 3)$ terminates and finally, top-2 communities found are h_1 and h_3 .

6.3 Keyword indexed tree exploration, TREE-EXPLORE

Though the above PRUNED-EXPLORE can prune a large number of subgraphs based on the derived upper bounds, it still explores subgraphs and their connected components with low cohesiveness, which usually do not contain the most influential communities. This exploration can be costly, especially in a scenario where the query essential graph, G_q , turns out to be very large. So, we propose a novel index, namely *keyword indexed core-label tree* (KIC-tree), that pre-computes and organizes the connected components of maximal k -core subgraphs hierarchically with computed upper bound of influence scores for each keyword.

The key idea of our KIC-tree based $KICQ$ comes from the following observations:

(i) Top communities are structurally cohesive and thereby can be retrieved by exploring the subgraphs of higher cohesion factors. Thus, if k -cores are precomputed, disregarding the associated keywords, we can still prune the subgraphs with low k value.

(ii) Communities are represented using connected maximal k -cores which are nested, i.e., by definition, a $(k + 1)$ -core is also a k -core ($k \geq 0$). This property helps to store all the connected components of maximal k -cores in compressed tree-based structures as shown in previous works ICP-index [7], CL-tree index [4].

(iii) We can compute the upper bounds for both the components: influence and cohesiveness, of the the scoring function, and use these upper bounds to prune the search space during query time.

We first discuss the basic structure of the KIC-tree index. Then we present the upper bounds for individual keywords and aggregate them for a set of keywords and predicates (in $KICQ$) for an upper bound score of a node. We also show how the cohesiveness score can be bounded based on a pre-computed structure alone. Finally, we present our TREE-EXPLORE algorithm for influential community search using the KIC-tree. In this section, we use the term “node” to exclusively indicate a tree node.

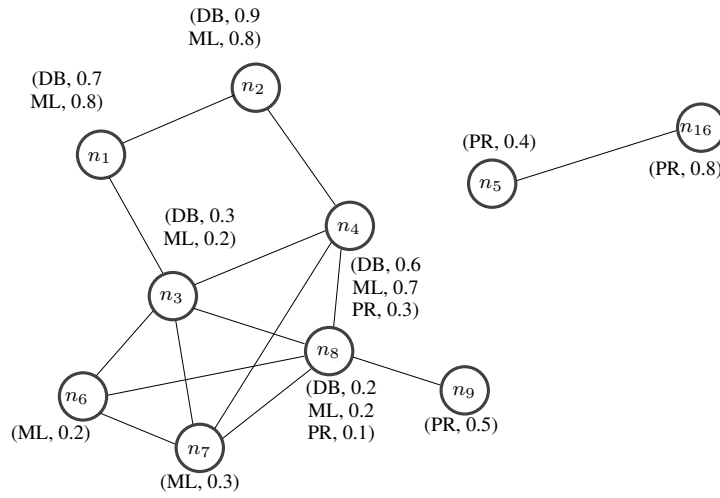


Figure 6.2: A subgraph of the graph presented in Figure 1.1.

6.3.1 KIC-tree index

The `KIC-tree` index organizes the connected components of k -cores into a space-efficient tree structure. We adopt the concept of compressed tree based structure of previous works (e.g., `CL-tree index` [4]), and augment the structure with derived bounds to prune the search space.

For simplicity of presentation, let us consider a smaller subgraph of the social network presented in Figure 1.1, which is shown in Figure 6.2. Figure 6.3 shows the corresponding `KIC-tree`. The left shows the hierarchical representation of all maximal k -core connected components in the subgraph. We refer this tree as the *uncompressed tree*. The right figure shows `KIC-tree` index, a more compact representation of the left tree, which removes the graph vertices present in its descendant nodes ensuring that each graph vertex appears exactly once.

Let u be a `KIC-tree` node and $subtree(u)$ be the subtree rooted at u . The structure of u is as follows:

- (i) k , the cohesion factor;
- (ii) $vertexSet$, the set of compressed graph vertices at node u ;
- (iii) $childNodes$, the set of child nodes of u ;
- (iv) k_{max} , the maximum cohesion factor of any connected component contained by the $subtree(u)$;

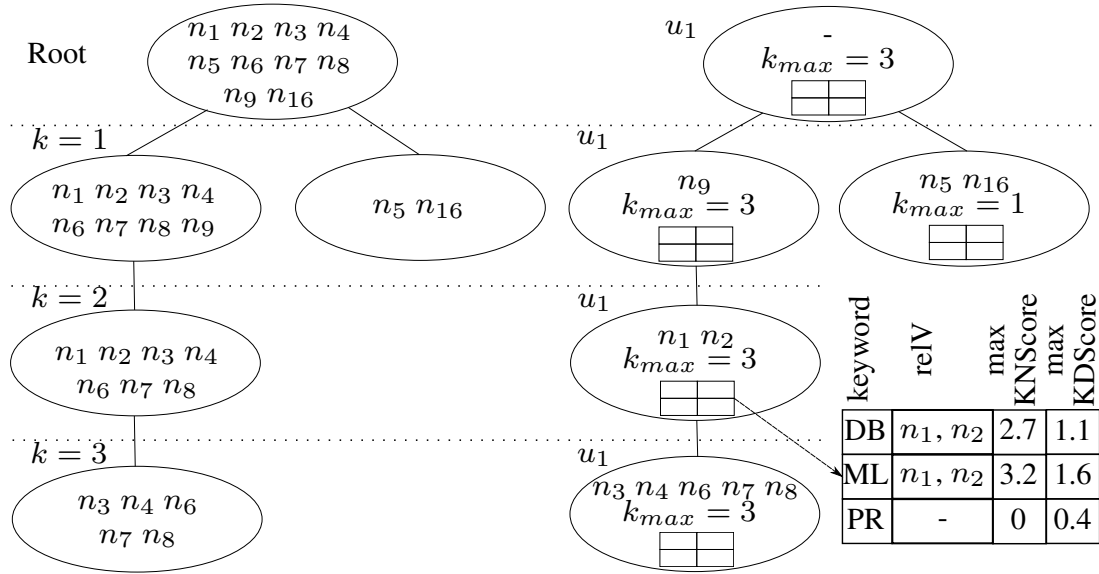


Figure 6.3: KIC-tree for the attributed graph in Figure 6.2

(v) $iList$, an inverted list containing the upper bounds of influence scores for all keywords appeared in $subtree(u)$.

For each keyword w that appears in $subtree(u)$, the inverted list $u.iList[w]$ contain the following elements:

- (i) $relV$, a set of graph vertices in $u.vertexSet$ containing the keyword w ;
- (ii) $maxKNScore$, the upper bound of influence score component by only considering keyword w in a community (i.e., a connected component) contained by the $subtree(u)$, where the community must include at least one vertex present in node u containing the keyword w ;
- (iii) $maxKDScore$, the upper bound of influence score component by only considering keyword w of a community contained by the $subtree(u)$, where the community does not include any vertex from $u.vertexSet$ (i.e., all vertices of the community come from the descendent nodes of u).

We compute $maxKNScore$ and $maxKDScore$ as follows.

For a node u , let $childV$ be the set of graph vertices stored at the descendent nodes of u , and $allV$ be the set of graph vertices at $subtree(u)$ (i.e., all the vertices in node u and its descendent nodes).

If u is a leaf node, $u.childV = \emptyset$.

Otherwise, $u.childV = \bigcup_{p \in u.childNodes} p.allV$. On the other hand, in all cases, $u.allV = u.vertexSet \cup u.childV$.

Now, if there is no relevant graph vertex in node u for keyword w , then we set $maxKNScore$ as 0. Otherwise, the upper bound is the sum of influence scores of all graph vertices in $u.allV$. Formally,

$$u.iList[w].maxKNScore = \begin{cases} 0, & \text{if } u.iList[w].relV = \emptyset \\ \sum_{v \in u.allV} (s_v(w)), & \text{otherwise} \end{cases}$$

Here, $s_v(w)$ is the influence score of vertex v for keyword w .

Now, $maxKDScore$ is the maximum influence score component among the communities represented by the descendant nodes of u .

$u.iList[w].maxKDScore = 0$ if u is a leaf node.

Otherwise, we can use the computed values of $maxKNScore$ to compute the $maxKDScore$ as follows.

$$u.iList[w].maxKDScore = \max_{p \in u.childNodes} \max(p.iList[w].maxKNScore, p.iList[w].maxKDScore)$$

Figure 6.3 (right) shows an example tree, where the table inside the ellipse represents the $iList$ of the corresponding node. For simplicity, we only show the $iList$ for node u_3 .

6.3.2 Complexity analysis for index construction

We use the advanced method proposed by Fang et. al. [4] that compresses the tree and for each node u , computes $u.iList[w].relV$ for all the relevant keywords of u . The time complexity of this method is $O(|E| \times \alpha(|V|))$, where $\alpha(|V|)$, the inverse Ackermann function, is less than 5 for all remotely practical values of $|V|$. For each $iList[w]$ entry, we also need to compute the two upper bounds $maxKNScore$ and $maxKDScore$. If A_{max} is the maximum number of keywords associated with a graph vertex, the time complexity for computing $maxKNScore$ is $O(A_{max} \times |V|)$. Computing $maxKDScore$ for a node u only requires visiting its $childNodes$ which is non-dominant. So, overall time complexity for index construction is $O(|E| \times \alpha(|V|) +$

$A_{max} \times |V|$).

In $iList$, we need additional space to store two upper bound scores (constant space) for each keyword. The space cost is still dominated by storing $relV$ in $iList$. So, the space complexity remains $O(\bar{A} \times |V|)$ as in [4], which is proportional to the graph size (\bar{A} is the average number of keywords associated with each vertex).

6.3.3 Computing upper bound scores for a query

Given a $KICQ(X, P, r, k_{min})$ query, we need to compute an upper bound influence score of a community denoted by S_{inf} and the maximum possible cohesiveness score of that community, S_k by using the precomputed upper bounds in $KIC-tree$. Then the upper bound of the total score of that community can be computed as $maxScore = \beta \times S_k + (1 - \beta) \times S_{inf}$ (as in Equation 5.2).

We define two upper bounds for the communities inside $subtree(u)$: (i) $maxNodeScore$, the maximum possible score of any community that can be exclusively found by exploring the connected k -core stored at node u and (ii) $maxDesScore$, the maximum possible score of any community that can be found by exploring the descendant nodes of u .

Computing $maxNodeScore$: For any community contained exclusively by node u , there must be at least one vertex v that is stored at u . Now, for any vertex v exclusive to node u , $u.k$ is the maximum core number. So, a subgraph containing v can be at most $u.k$ -core (irrespective of any keyword) and the upper bound of cohesiveness score of any community contained by the node can be computed as $S_k = u.k / max-deg(G^+)$.

Now, for each keyword w , $u.iList[w].maxKNScore$ already defines the upper bound of influence score component for any community in the subgraph exclusively contained by node u (considering the single keyword w). We combine these bounds for considering all the keywords in the $KICQ$ query and compute the maximum influence score as:

$$S_{inf} = \frac{1}{|V|} \times \mathbf{F}_{X_{t_i} \in X} (\sum_{w \in X_{t_i}} u.iList[w].maxKNScore)$$

Here \mathbf{F} is an aggregate function that combines the influence scores of the community for multiple terms depending on the predicate P and division by $|V|$ normalizes the score within

[0,1]. For the queries with OR predicate, a top community can be formed by joining multiple communities pre-computed for a single term, and these communities may have disjoint vertex set. So, it is safe to consider \mathbf{F} as a **summation** aggregate. For the same reason, \sum is explicitly used to combine the semantic keywords of a term. Again, for the queries with AND predicate, any graph vertex forming a community for a single term must be present in communities of other terms as well. So, \mathbf{F} can be safely considered as **minimum** aggregate.

Computing $maxDesScore$: For any community contained by the descendant nodes of u , the maximum cohesion factor is $u.k_{max}$ and the upper bound of cohesiveness score is $S_k = u.k_{max}/max-deg(G^+)$.

Again, for keyword w , $u.iList[w].maxKDScore$ already defines the upper bound of influence score component for any community contained by the descendant nodes. We combine these bounds for considering all the keywords in the $KICQ$ query and compute the maximum influence score similarly as computing $maxNodeScore$, i.e.,

$$S_{inf} = \frac{1}{|V|} \times \mathbf{F}_{X_{t_i} \in X} (\sum_{w \in X_{t_i}} u.iList[w].maxKDScore)$$

6.3.4 TREE-EXPLORE algorithm

We follow a *best-first* exploration strategy. Since the leaf nodes contain the communities with high cohesiveness while nodes near root contain communities with low cohesiveness, we explore the $KIC-tree$ in a post-order manner. Likewise the previous exploration algorithms (e.g., $PRUNED-EXPLORE$), a priority queue Q initialized with r empty communities is used to store the results. The exploration algorithm, which we call $TREE-EXPLORE$ is developed based on the following pruning techniques:

(i) **Subtree pruning:** For any node u , we examine the $u.maxDesScore$ before visiting its children. If it is less than the r^{th} best score, then none of the communities to be found in the descendent nodes can score higher than the current r^{th} top community. Therefore, we can skip visiting the descendant nodes of u .

(ii) **Node pruning:** Before exploring the pre-computed connected k -core subgraph at any node u , we examine the $u.maxNodeScore$. If it is less than the r^{th} best score, we can safely

Algorithm 3 TREE-EXPLORE (u, U)**Input:** Tree node u , query relevant nodes U .

```

1: if  $u$  is an internal node then
2:   Compute influence score component  $S_{inf}$  and cohesiveness score component  $S_k$  for  $u.maxDesScore$ 
3:    $u.maxDesScore = \beta \times S_k + (1 - \beta) \times S_{inf}$ 
4:   if  $S_{inf} > 0$  and  $u.maxDesScore > r^{th}$  best score then
5:     for each  $p \in (u.childNodes \cap U)$  do
6:       TREE-EXPLORE( $p, U$ )
7: if  $u.k < k_{min}$  then
8:   return
9: Compute influence score component  $S_{inf}$  and cohesiveness score component  $S_k$  for  $u.maxNodeScore$ 
10:  $u.maxNodeScore = \beta \times S_k + (1 - \beta) \times S_{inf}$ 
11: if  $S_{inf} = 0$  or  $u.maxNodeScore < r^{th}$  best score then
12:   return
13:  $u.V_{rel} =$  compute relevant graph vertices in the subtree rooted at  $u$ 
14: Compute query relevance score of all vertices in  $u.V_{rel}$ 
15: Compute  $u.E_{rel}$ , the edges among  $u.V_{rel}$ 
16: Construct subgraph  $H(u.V_{rel}, u.E_{rel})$ , each vertex annotated with relevance score
17: MODIFIED-PRUNED-EXPLORE( $H, k_{min}, u.k$ )

```

prune this exploration.

Algorithm 3 outlines the pseudocode for the KIC-tree traversal. The inverted list that we have used to find the G_q is adopted for computing U , the set of tree nodes relevant to a query. Initially, the recursive procedure TREE-EXPLORE(u, U) is called with u being the root of the KIC-tree.

For any internal node u , we first compute and examine the influence score component, S_{inf} , and the cohesiveness score component, S_k of $u.maxDesScore$. If S_{inf} is 0, the descendants of u do not contain any graph vertex relevant to the query, and therefore we do not need to visit subsequent nodes across the subtree. If $S_{inf} > 0$ and $u.maxDesScore$ is greater than the current r^{th} best score, then we visit its children (lines 1-6).

Now we want to explore the pre-computed connected k -core subgraph represented by node u . If the cohesion factor k in node u is less than k_{min} , then we prune exploring the subgraph. Otherwise, we compute the influence score component (S_{inf}) and the cohesiveness score component (S_k) of $u.maxNodeScore$. If S_{inf} is 0, then the node does not contain any graph vertex relevant to the query, and we can safely skip exploring the subgraph. Again, we skip the exploration if $u.maxNodeScore$ is less than the current r^{th} best score (lines 7-12).

If the exploration of the connected k -core subgraph cannot be pruned, we first need to find all the relevant graph vertices, $u.V_{rel}$ present in the subgraph (line 13). Since KIC-tree

compresses these graph vertices by removing ones present at descendant nodes, we need to decompress in a bottom-up manner. At any node u , the relevant graph vertices $u.V_{rel}$ can be computed like the vertices in QEG (Equation 6.1) just by replacing IL_w with $u.iList[w].relV$. For any internal node u , we need to add the relevant graph vertices in child nodes to $u.V_{rel}$.

Now we compute the relevance score of each vertex $v \in u.V_{rel}$ (Equation 5.1) and then compute the edges among these vertices, thereby construct the subgraph $H(u.V_{rel}, u.E_{rel})$ (lines 14-16).

The procedure $\text{MODIFIED-PRUNED-EXPLORE}(H, k, k_{max})$ is a slightly modified version of the procedure $\text{PRUNED-EXPLORE}(H, k)$ that takes an extra argument k_{max} , the maximum value of cohesion factor for sub-graph H . Lines 10-13 in Algorithm 2 are replaced by the following:

```

10: for  $k' = k + 1$  to  $k_{max}$  do
11:   if  $\zeta_{k'}^*(h) > r^{th}$  best score then
12:      $\text{MODIFIED-PRUNED-EXPLORE}(h, k', k_{max})$ 
13:     break

```

Since no graph vertex at u belongs to any k -core with cohesion factor higher than $u.k$, here $k_{max} = u.k$. Initially, $k = k_{min}$. So, $\text{MODIFIED-PRUNED-EXPLORE}(H, k_{min}, u.k)$ is called to explore the subgraph H (line 17).

Chapter 7

Experimental Study

In this chapter, we present experiments to evaluate the performance of our proposed system. We first describe the experimental setup, then evaluate our semantic similarity model, and finally, we conduct experiments to evaluate the efficiency and effectiveness of our proposed influential community search algorithms.

7.1 Experimental setup

We implemented the semantic similarity model in Python using nltk and gensim libraries. All the community search algorithms are implemented in JAVA. Experiments were run on a virtual environment of OzSTAR¹ supercomputer with two cores of Intel Gold 6140 CPU @ 2.30 GHz 2.30GHz, 192 GB RAM, and 400 GB SSD. We assume that the graph and all the indexes will fit in the memory. For the simplicity of presentation, we use shorter names for our algorithms: BASIC, PRUNE, and TREE to represent BASIC-EXPLORE, PRUNED-EXPLORE, and TREE-EXPLORE respectively. We present the average results of 100 queries.

¹ <https://supercomputing.swin.edu.au/ozstar/>

7.1.1 Datasets

We use two large real datasets: OAG (Open Academic Graph)² [55] and Gowalla³ to generate the attributed graph reflecting the real life application scenarios. Both of the computed attributed graphs are publicly available⁴.

OAG is generated by linking two large academic graphs: Microsoft Academic Graph (MAG) and AMiner. Here, we represent each author as a vertex and co-authorships between authors as edges. This dataset contains more than 150 million academic articles with metadata like title, abstract, authors, keywords provided by authors, etc. We first choose 1,000 most frequent author-provided keywords as the set of keywords for the attributed graph. Each vertex of the graph is further augmented with the semantically relevant set of keywords (by applying our semantic similarity model) for the author provided keywords. The score is modeled as the author's percentile rank (scaled to 1.0), considering the number of citations in publications relevant to a keyword. We skipped non-English articles and the articles with no citation. Finally, there were 10,714,737 articles in our dataset. In the attributed graph, we considered the first 1 million authors as vertices and 15,677,940 co-authorship relations among the authors as edges.

In Gowalla dataset, users are modeled as vertices, and the location ids are considered as keywords. Edges represent the friendship between two users. Each user is augmented with the locations where she checked in. The influence score of a user at a location is modeled as the user's percentile rank considering the number of check-ins posted by the user at that location. There are 407,533 vertices, 2,209,169 edges and 2,727,464 keywords in this attributed graph.

7.1.2 Query Generation

To generate a query for OAG datasets, first, we choose the number of query terms randomly within [1,5]. Then we randomly choose a keyword from the most frequent 10,000 author-provided keywords. This keyword is taken as the first query term. We choose the rest of the query terms randomly. Any keyword that appears with the first query term in the author-provided

² <https://aminer.org/open-academic-graph>

³ <http://www.yongliu.org/datasets/index.html>

⁴ <https://drive.google.com/drive/folders/1yU32kH6E2xvQow8hxCbCtMoFASDVn4TE?usp=sharing>

keyword list is a candidate to be chosen as a query term. Additionally, we augment each keyword with its semantically similar keywords using our `semantic keyword similarity` model. For each set of query terms, we take both AND and OR predicates. Note that low-frequency keywords are usually very specific, and when we consider AND predicate among them, the communities may not be meaningful. For example, queries like “**astrophysics**” AND “**genomics**” do not yield meaningful communities. For this reason, we do not choose straightforward random keywords as a query. Parameters r and k_{min} are set to default values unless otherwise specified.

For Gowalla dataset, we randomly choose a location as the first keyword, and then choose 0-4 additional random locations within the radius of $5km$ from the first location. We do so because if the locations are very far from each other, there will be very few users who visit all the locations, thereby making the AND queries meaningless. For the entire query setup, we only consider the locations where at least 50 users checked in.

7.1.3 Experiment parameters

We vary different parameters as shown in Table 7.1. When one parameter is varied, other parameters are fixed at their default values.

Parameter	Range	Default
Dataset	OAG, Gowalla	OAG
Dataset size (vertices)	300K, 500K, 700K, 900K, 1M	500K
Number of keywords	100, 250, 500, 750, 1000	1000
β	any real value within [0.0, 1.0]	0.60
r	any integer within [1, 5]	3
k_{min}	any integer within [2, 50]	10

Table 7.1: Parameters for experimental analysis

7.2 Evaluation of semantic similarity model

In this section, we first present the experimental studies to evaluate our proposed similarity metrics followed by parameter selection for the semantic similarity model.

7.2.1 Selection of similarity metric

In Chapter 4, we presented three approaches for finding semantic similarity between two terms or keywords. Here, we empirically evaluate which approach is the most effective. Our goal here is to measure a similarity between two terms using the three measures, and compare the similarity results with the ground truth. As the ground truth, we use the semantic similarity measure that has been widely used in Information Retrieval for measuring the similarity between two concepts in a given taxonomy [56]. Its unique feature is measuring the similarity based on the intrinsic information content of the taxonomy structure only. As the taxonomy, we use the ACM taxonomy provided by the 2012 ACM Computing Classification System⁵ that contains 2,113 topics and organizes them hierarchically based on relevance (e.g., “clustering” is a sub-topic of “data mining”).

In our context, each topic in the taxonomy can be seen as a keyword or a term, and each of our proposed similarity measures can be considered as a ranker that finds the most similar topics to any topic in the taxonomy. So, we use a widely accepted performance measure of ranking systems, that is, Normalized Discounted Cumulative Gain (*NDCG*) [57], to evaluate the proposed similarity metrics.

First, given the taxonomy τ , we give the ground truth formula of the similarity between two topics t_1 and t_2 proposed in [56]:

$$sim_{jcn}(t_1, t_2) = 1 - [d_{jcn}^\tau(t_1, t_2)/2]$$

⁵ https://dl.acm.org/ccs/ccs_flat.cfm

Here, $d_{jcn}^\tau(t_1, t_2)$ denotes a distance metric between t_1 and t_2 , defined by Jiang et al. [58].

$$d_{jcn}^\tau(t_1, t_2) = IC^\tau(t_1) + IC^\tau(t_2) - 2 \times IC^\tau(lcs(t_1, t_2))$$

$lcs(t_1, t_2)$ is the “least common subsumer” in τ that subsumes t_1 and t_2 . Finally, $IC^\tau(t)$ indicates the information content of topic t in τ , calculated by $\log(\frac{|sc^\tau(t)|+1}{|\tau|}) / \log(\frac{1}{|\tau|})$ as in [56]. Here, $sc^\tau(t)$ is the set of subsumed topics of t and $|\tau|$ is the total number of topics in τ . The denominator $\log(\frac{1}{|\tau|})$, which is equivalent to the value of the most informative topic, serves as a normalizing factor, assuring that the information content values are in $[0,1]$.

Second, given a topic t in τ , our task is to rank the M -top similar topics $\{w_1, w_2, \dots, w_M\}$ in τ . The ranking goodness is evaluated by $NDCG@M$ defined as

$$NDCG = \frac{1}{|\tau|} \sum_{t \in \tau} NDCG^t \quad (7.1)$$

Here, $NDCG^t = \frac{DCG^t}{IDCG^t}$ where, $DCG^t = \sum_{i=1}^M \frac{sim_{jcn}(t, w_i)}{\log(i+1)}$ is the discounted cumulative gain that gives more importance on the ranking of a more relevant entity than the ranking of entities with lower relevance. $IDCG^t$ provides the maximum DCG value that could be obtained by a ranker if the ranking was ideal.

To obtain a vector representation of each topic in τ , we used Google’s pre-trained word2vec model⁶. This model includes embedding vectors for a vocabulary of 3 million words and is trained on about 100 billion words from a Google News dataset which also covers academic research. The length of the embedding vector is 300.

Table 7.2 shows the comparison among the three similarity metrics in terms of $NDCG@50$, $NDCG@20$, and $NDCG@10$. As seen, overall, it turns out that `indirect cosine` outperforms the other metrics. Thereby, we have chosen this metric to associate each term or keyword with its semantically related keywords or terms.

⁶ <https://code.google.com/archive/p/word2vec/>

Metric	cosine	normalized maximum	indirect cosine
NDCG@50	0.528	0.543	0.542
NDCG@20	0.537	0.583	0.607
NDCG@10	0.573	0.614	0.633

Table 7.2: Performance results of the three similarity metrics

7.2.2 Parameter setting

In the `indirect cosine` similarity measure, given a term t , we need to retrieve its semantically relevant vector $V^t = [(w_1^t, s_1^t), (w_2^t, s_2^t), \dots, (w_L^t, s_L^t)]$ (see Chapter 4). V^t contains L -top words and their similarity to t . Here, a challenge is how to determine a good value for L . To examine this, we use three measures.

The first is *word coherence*, $WC^L(V^t)$, to evaluate how coherent the L -top words in vector V^t are. The more coherent it is, the better we can represent the set of keywords. We define the word coherence as the average pairwise cosine similarity of the L -top words. Formally,

$$WC^L(V^t) = \frac{2}{L \times (L - 1)} \sum_{i=1}^L \sum_{j=i+1}^L sim(w_i, w_j) \quad (7.2)$$

Here $sim(w_i, w_j)$ is the cosine similarity of the embedding vectors of w_i and w_j . $WC^L = 0$ indicates no coherence and value $WC^L = 1$ indicates maximum coherence.

Also, in a sense, each term is a cluster containing its relevant words. As the second measure, we use *Davies-Bouldin Index* [59] that optimizes two criteria: (1) minimizing intra-distance between words and the centroid, and (2) maximizing inter-distance between keywords. Values closer to zero indicate a better clustering.

Again, the number of similar keywords found depends on the value of L . For example, when $L = 2$, a keyword will be considered similar to a given term only if at least one of the two top words matches. If $L = 50$, the keyword will be somewhat similar to the term if they have at least one top word common among the 50 top words. Let, we are to retrieve M most similar keywords to a given term t . If we are able to retrieve m keywords, then the percentage of top- M retrieval

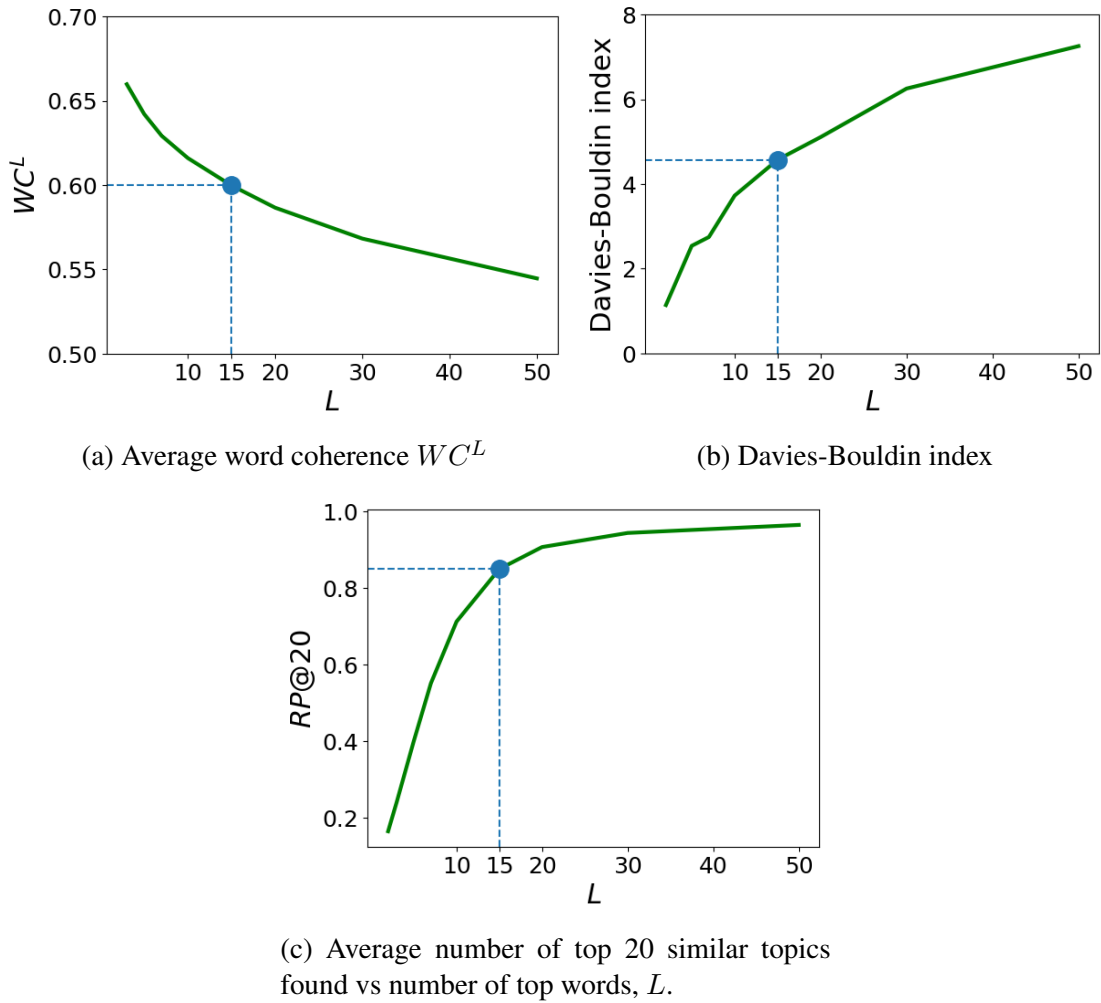


Figure 7.1: $L=15$ (L : number of top words) provides reasonably high keyword coherence, low Davies-Bouldin index, and high retrieval percentage

is defined as

$$RP@M^t = \frac{m}{M} \quad (7.3)$$

Average percentage of top- M retrieval, $RP@M$ is calculated by averaging $RP@M^t$ over all terms.

Small value of L usually contains words with high coherence and better clustering, but it is very difficult to find similar keywords. Again, large value of L contains words with low coherence and worse clustering, but it is easier to find similar keywords. So, we need to choose a value which provides a good trade-off. The desired value of L should provide high word coherence WC^L , low Davies-Bouldin index, and high retrieval percentage $RP@M$. Figure 7.1

shows the average cohesiveness, Davies-Bouldin index, average percentage of top 20 retrieval vs various values of L . Here $L = 15$ provides a good trade-off.

7.3 Evaluation of KICQ

The OAG dataset is enriched with metadata from millions of articles, and better fits the application scenarios of our study, and thus we use it as the default dataset, unless otherwise stated, to demonstrate the performance of our proposed algorithms. First, we evaluate how different parameters (Table 7.1) affect the efficiency and effectiveness of the competitive algorithms. Then we compare the performance of our algorithm with the state-of-the-art influential community search algorithm [7].

7.3.1 Performance evaluation

In this section, we show the scalability, sensitivity of different parameters, memory requirement, and the cohesiveness of the retrieved communities by running a wide range of experiments.

KICQ processing time

In this set of experiments, we evaluate and compare the runtime of our proposed algorithms.

Varying the dataset size: To show the scalability, we consider different size of OAG dataset by varying the number of vertices and thereby edges. Figure 7.2 shows that with the increase in the number of vertices, runtime also increases. Usually, when AND predicates are involved, most of the vertices become irrelevant to the query and get filtered out making the query essential graph is much smaller than OR predicates. So, OR query is more time consuming and challenging. We can see that TREE-EXPLORE significantly outperforms the other approaches for OR queries with various number of keywords. For AND query, the advantage of pruning few vertices is ruled out by the overhead of exploring the large tree in TREE-EXPLORE, but it still maintains reasonable performance. Also, PRUNED-EXPLORE and TREE-EXPLORE scales much better than BASIC-EXPLORE.

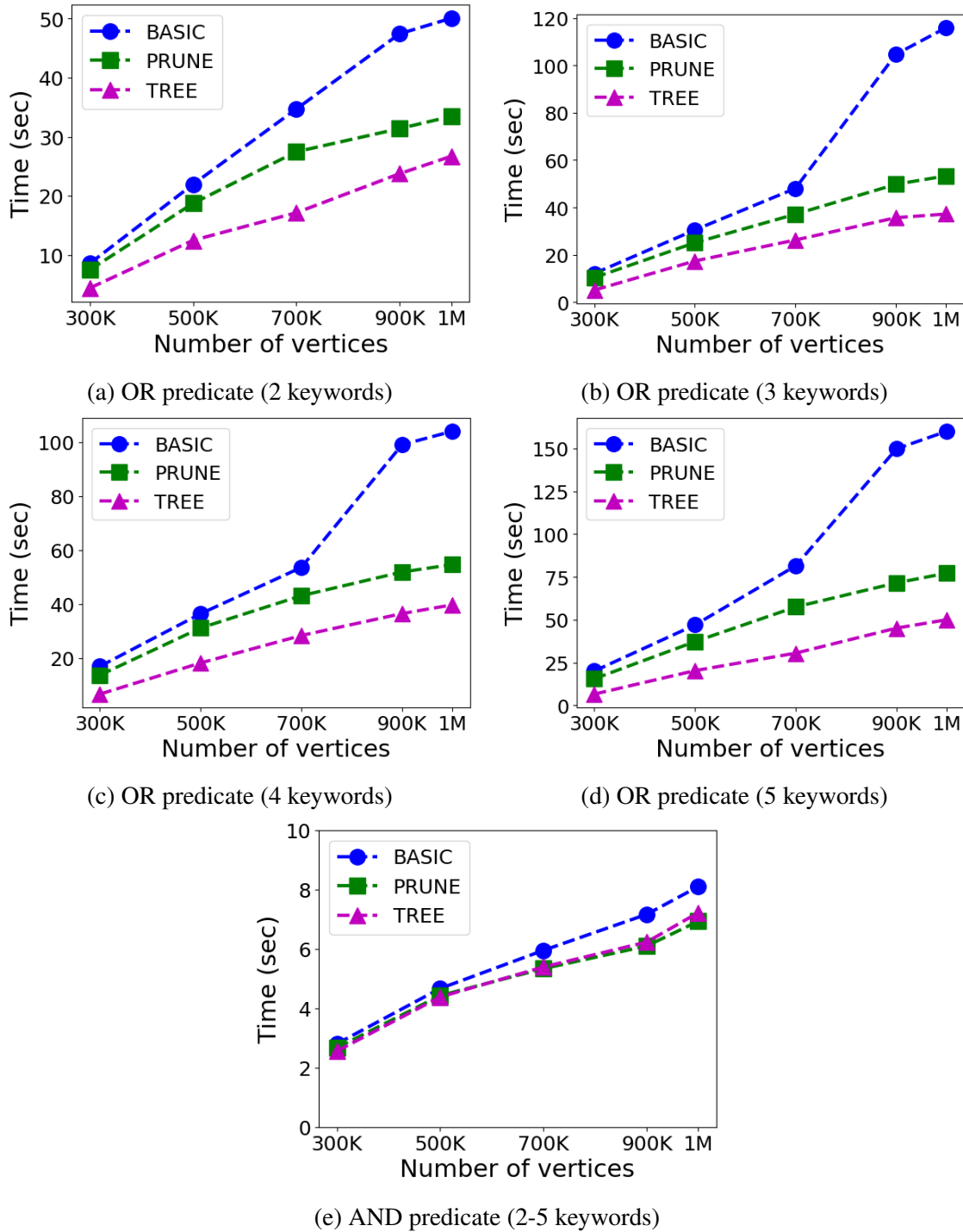


Figure 7.2: Query processing time for varying dataset size

Varying k_{min} . Figure 7.3 shows how the query processing time is affected by parameter k_{min} . None of the BASIC-EXPLORE and PRUNED-EXPLORE algorithms are significantly affected

by the value of k_{min} . However, if k_{min} is set to a high value, TREE-EXPLORE does not need to explore the tree nodes representing k -cores with lower k values. This enables TREE-EXPLORE to skip a larger part of the tree since most of the vertices in the OAG dataset has degree less than 10 making TREE-EXPLORE significantly faster for higher k_{min} values.

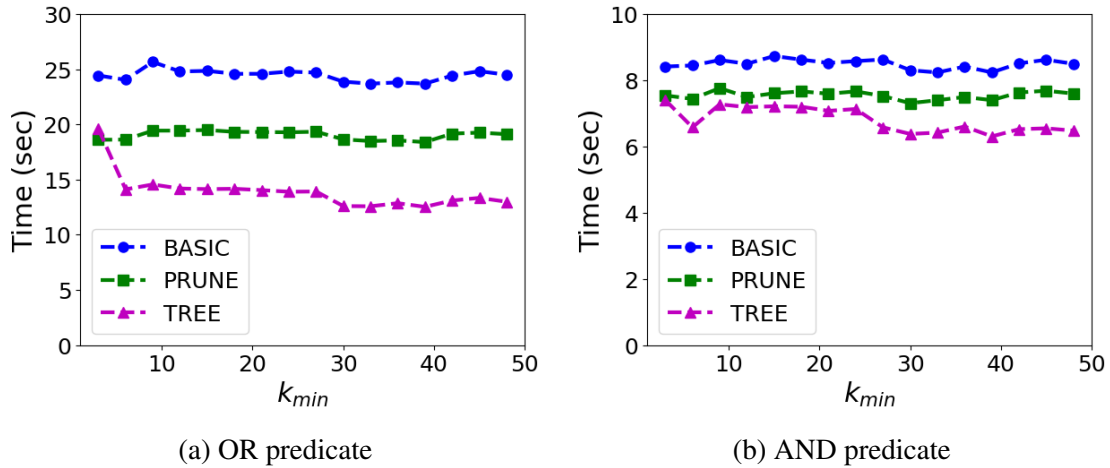


Figure 7.3: Query processing time for varying k_{min}

Varying r . Figure 7.4 demonstrates how the query processing time is affected by the number of communities to be retrieved (query parameter r). Both BASIC-EXPLORE and PRUNED-EXPLORE compute core decomposition once on the entire query essential graph. However, TREE-EXPLORE performs the core decomposition on-demand basis. BASIC-EXPLORE is not affected by the value of r since it does not use any pruning based on the retrieved communities. PRUNED-EXPLORE prunes some expansion based on top- r score, but the query processing time is not noticeably affected. The effect is more substantial in TREE-EXPLORE. A significant part of the graph does not require core decomposition if r is small. If r is high, the algorithm can only prune a few tree nodes, but the advantage is ruled out by the overhead of decompressing the graph vertices inside a tree node. However, for small values of r , TREE-EXPLORE significantly outperforms the other algorithms, especially for OR predicate where query processing time is markedly higher than the AND predicate.

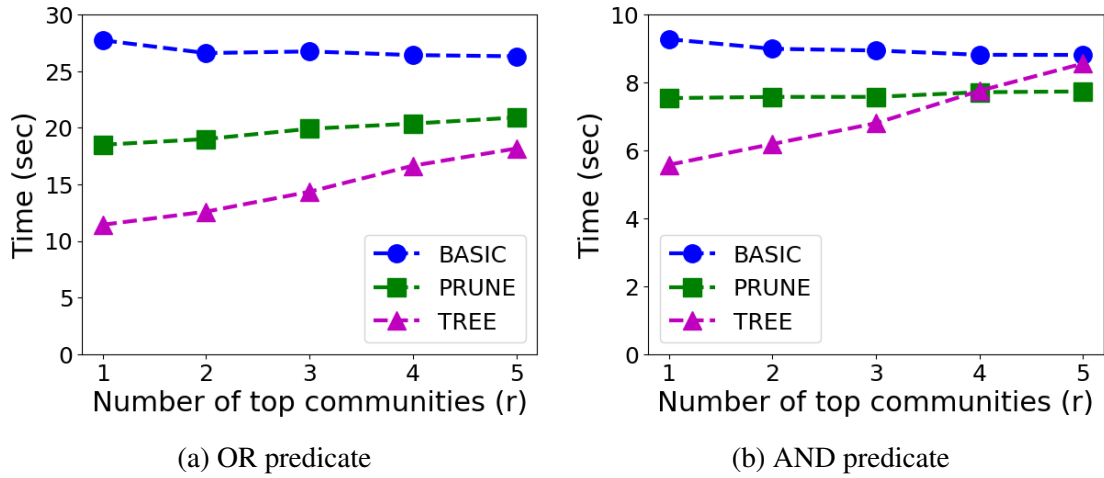


Figure 7.4: Query processing time for varying r

Index size

Figure 7.5 shows how the size of the graph and corresponding `KIC-tree` index increase with the increasing number of vertices and keywords. The result conforms to the complexity analysis demonstrated in Section 6.3.2. Index size is linear to both the number of vertices and number of keywords if one of these remains unchanged. Also, index size is bounded by the graph size.

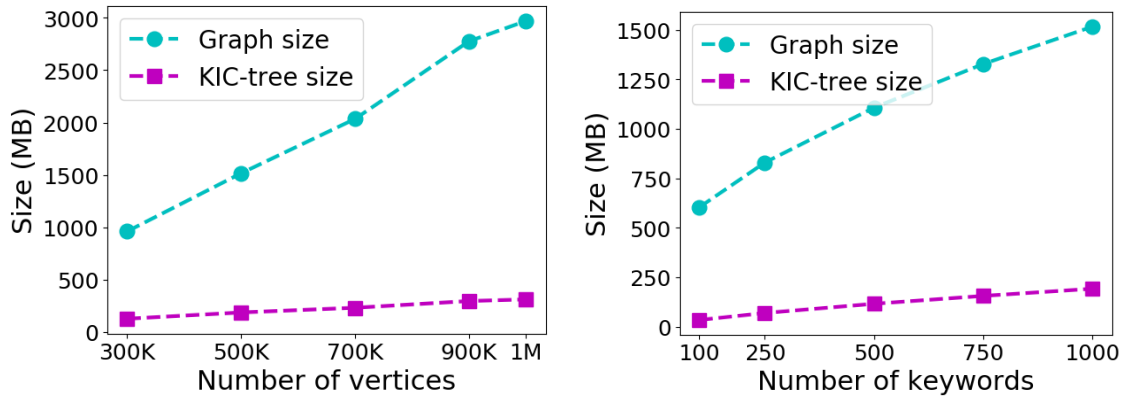


Figure 7.5: Index size for varying vertices and keywords

Structural cohesiveness

We use popular structural cohesiveness metrics diameter, density, average degree, and clustering coefficient [60] to measure the quality of the communities retrieved by our approach. These measures mostly depend on the community models (e.g., k -core, k -truss) as discussed in a survey of community search [60]. They prefer the k -core model because of its high efficiency with minimal sacrifice in structural cohesiveness. By analyzing the cohesiveness measures in Table 7.3, we claim that our algorithms can retrieve cohesive communities with a small diameter.

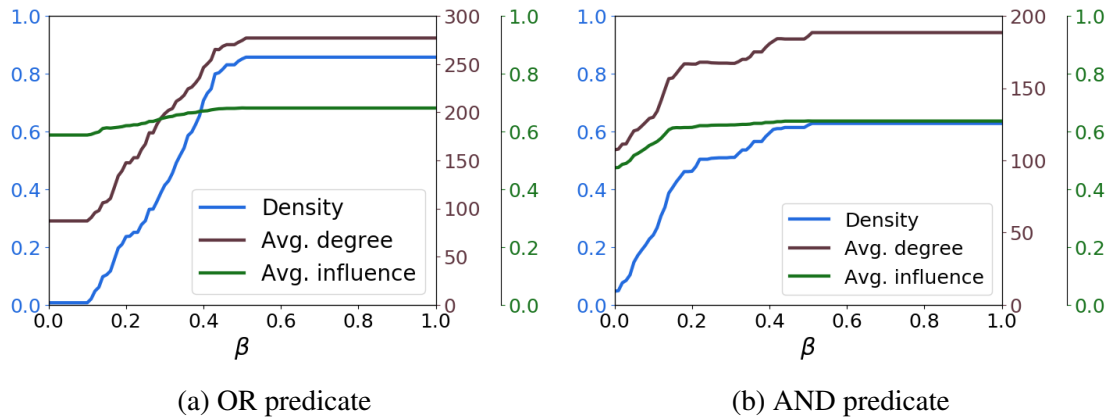
Dataset	Density	Average Degree	Clustering Coefficient	Diameter
OAG	0.621	177.897	0.708	2.12
Gowalla	0.579	7.484	0.881	2.70

Table 7.3: Structural cohesiveness measures

Setting the value of β

We first run experiments to find an appropriate β , which balances the weight of connectivity and individual influence (query relevance score). For larger β , top communities are likely to show more cohesiveness and discard communities with influential individuals but less connectivity. When β is smaller, the top communities might incline to individual influence than cohesiveness. To find a good choice of β , we consider the network structure, that is: (1) we plot structural cohesiveness measures (i.e., density and average degree [4]), and (2) use the average influence score of the members for different β values as shown in Figure 7.6.

For OAG, communities with higher cohesiveness seem to contain influential individuals. This can be easily explained by the fact that an author who has co-authorship with a large number of authors is likely to have a strong influence in her field of studies. So, for OAG, we choose a high value of β (i.e., 0.6). Note that, considering the influence of communities is still essential since it is the tie-breaker between two communities with the same cohesiveness.

Figure 7.6: Performance measures for varying β values

7.3.2 Comparison with state-of-the-art

We choose `Online-All` [7] as the state-of-the-art approach as they find influential community in non-attributed graphs.

Efficiency

To compare our algorithms with `Online-All`, we construct queries with a single keyword (as it does not support keywords) and compute the query essential graph, which is the input graph to `Online-All`. We also need to input the cohesiveness parameter k in `Online-All`. For this, we only consider the top community ($r = 1$), and the value of k in the top community returned by our approach is fed to `Online-All`. We do not consider the other algorithms in [7] since they use pre-computation which cannot be adopted for our problem. For fair comparison, we do not consider `TREE-EXPLORE` algorithm since it uses pre-computed index. We show the query processing times of these algorithms in Figure 7.7a. Our approaches are *four times* faster compared to the `Online-All` algorithm.

Cohesiveness

To compare the keyword cohesiveness, we use community pair-wise Jaccard (CPJ) metric proposed in [4]. CPJ measures the similarity of the members of top communities in terms of keywords. We form the queries as described in Section 7.1 to evaluate our approach. `OnlineAll` cannot process queries with keywords; rather, it requires a cohesiveness parameter k as input. For

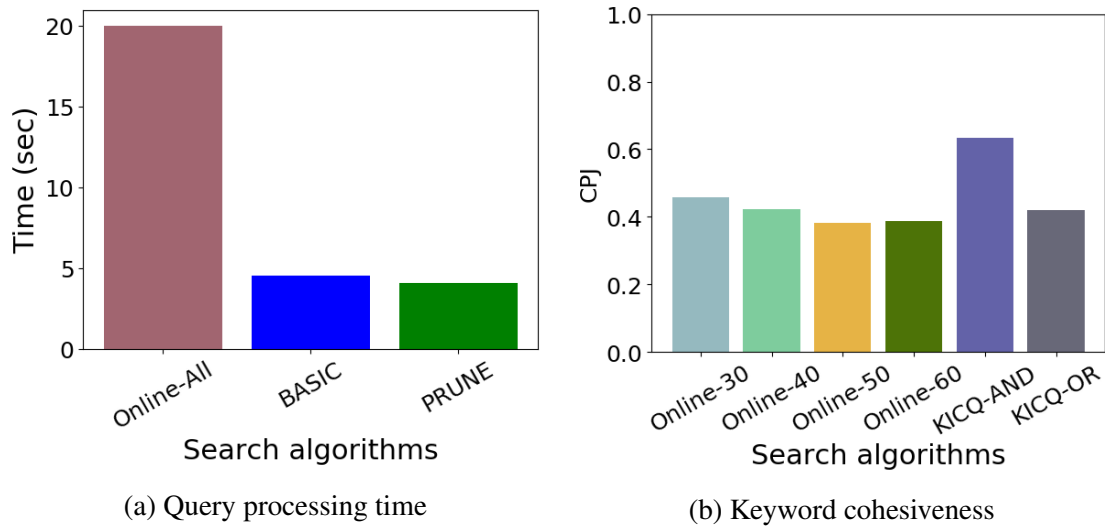


Figure 7.7: Effectiveness and efficiency comparison

the simplicity of presentation, we denote our approach as KICQ-AND, KICQ-OR for AND, OR predicates, respectively. `Online- x` denotes `OnlineAll` with parameter $k = x$. The comparison is presented in Figure 7.7b. For the communities returned by our approach with AND predicate, keyword cohesiveness is 1.5 times higher than `OnlineAll`, while for OR predicate CPI is similar. This is expected as for OAG, two vertices are only connected when corresponding authors publish a paper together, and they also share common keywords. For this reason, `OnlineAll` finds communities with good CPI value. However, this might not be the case for other social networks (e.g., Gowalla, Twitter). Note that our approach and [7] use the same community model (i.e., k -core), and the structural cohesiveness is similar.

7.4 Experiments with Gowalla dataset

We conduct experiments on Gowalla dataset to show that our algorithms can handle large number (millions) of keywords. For these experiments, we carefully craft the queries as described in Section 7.1, and use $\beta = 0.5$, $r = 3$, $k_{min} = 5$ as default values. First, we report the cohesiveness measures in Table 7.3 and observe that our approach can retrieve cohesive communities with a small diameter for Gowalla dataset as well. We also compare the effectiveness and efficiency of the communities retrieved by our approach, and Li et al. [7]. The query setup and

parameter settings are done in the same way as in the OAG dataset. The query processing time and keyword cohesiveness of these two approaches are presented in Figure 7.8. The number of relevant vertices is very small for queries in this dataset. The average query processing time is around 2 ms for all the algorithms. However, our key observation is that unlike OAG, in a dataset where connectivity is not related to keywords, the `Online-All` fails to address the keyword cohesiveness of the communities, while our approach returns communities considering both structural and keyword cohesiveness. The results show that our approach returns communities with significantly higher (approx. 15 and 5 times for AND and OR predicate respectively) keyword cohesiveness than the `Online-All`.

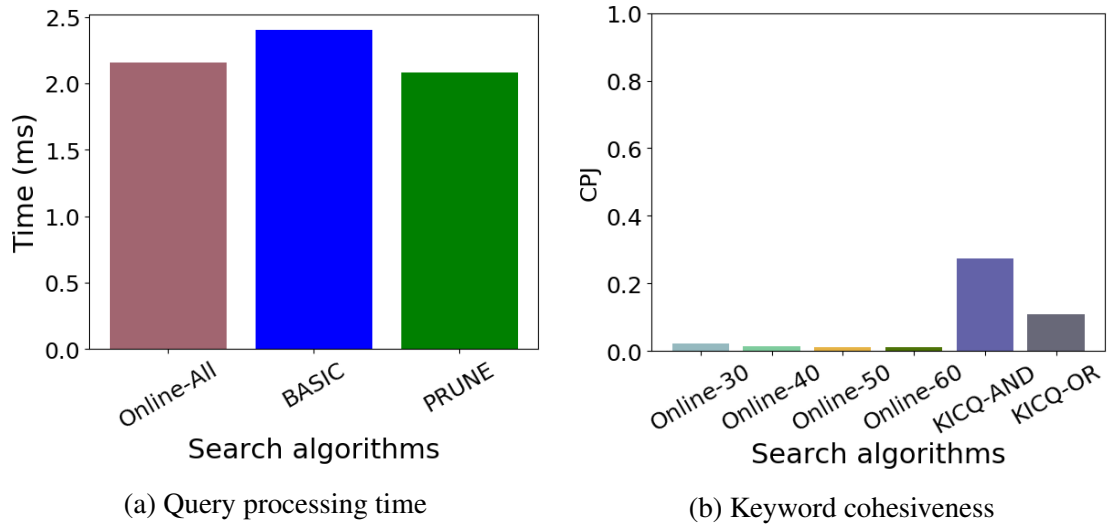


Figure 7.8: Effectiveness and efficiency comparison

Chapter 8

Case Study

We use a small dataset of co-author network from ArnetMiner¹ [55] to study the quality of retrieved communities. The dataset contains 5,411 vertices and 17,477 edges. Each vertex represents an author annotated with fields from eight different research areas: Data Mining (DM), Web Services (WS), Bayesian Networks (BN), Web Mining (WM), Semantic Web (SW), Machine Learning (ML), Database Systems (DS), and Information Retrieval (IR), where the influence score in each field depends on the number of publications in that field. There is an edge between two authors if they publish at least two papers together.

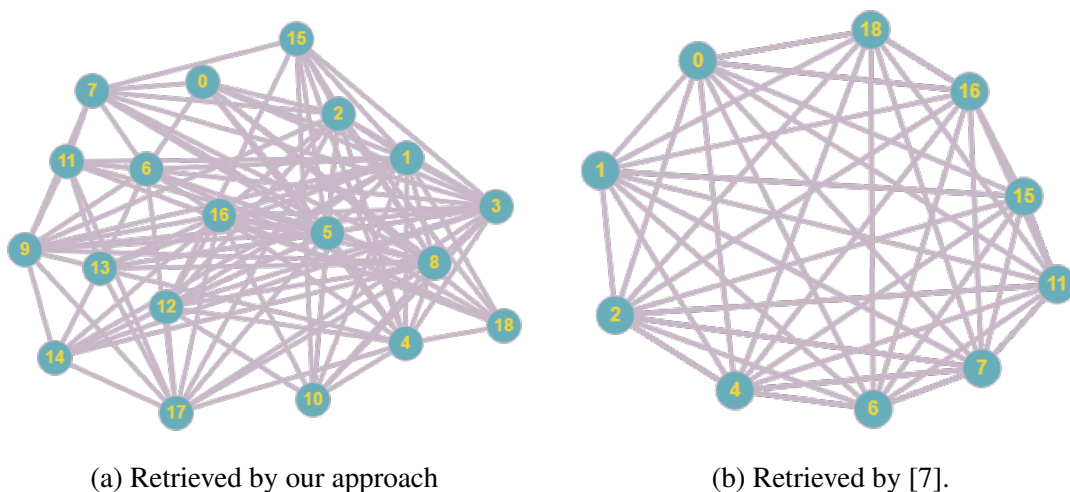


Figure 8.1: Retrieved top communities for DS.

¹ <https://aminer.org/lab-datasets/soinf/>

Note that [7] also conducted a case study on this dataset. Figure 8.1 shows the top community in DS retrieved by our approach (Figure 8.1a) and by [7] (Figure 8.1b). The top community returned by our approach is 8-core and thus we compare the result of [7] for $k = 8$. The details, i.e., h-index and number of citations of each author in our community are shown in Table 8.1. Among them, the authors who are not included in [7]’s community are shown in bold text. Due to the minimum score modelling, they missed out some of the top authors in this area including Rakesh Agrawal who was awarded the most influential scholar in the research area of Database Systems (DS) in Aminer ². Our approach keeps him in the community as we did not exclude a relatively less influential (with good connectivity) author Laura M. Haas. When Laura is not included in the community, Rakesh Agarwal is connected to less than 8 authors in the community, which turns out to be a non 8-core community. Since in the minimum weight modelling of [7], inclusion of a low influential member like Laura M. Haas significantly reduces the score of the entire community, the resultant community no longer remains the top community for $k = 8$. These findings show the effectiveness of our problem formulation and score function modelling.

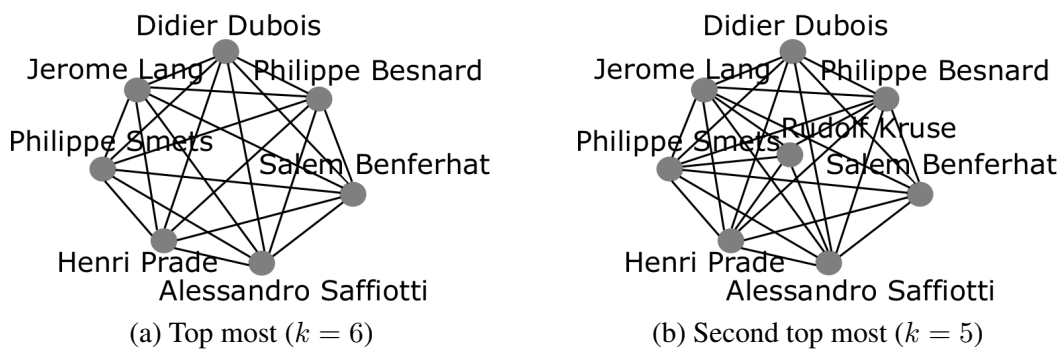


Figure 8.2: Top communities for BN OR DM.

Figure 8.2 presents two top communities for “BN OR DM” returned by our algorithms. The top most community is fully connected and contains highly influential authors like Didier Dubois (h-index: 125, citations: 82,295), Henri Prade (h-index: 119, citations: 78,700). The second top most community also contains all the authors from top-1 community, but the inclusion of another author Rudolf Kruse (h-index: 54, citations: 16,829) increases the contribution of individual scores, but decreases the cohesiveness of the community resulting in a lower total score than

² <https://aminer.org/mostinfluentialscholar>

the first one. This shows the flexibility and the trade-off capability among parameters while searching for the communities.

Vertex Id	Author Name	h-index	Citations
0	Hector Garcia-Molina	138	90,220
1	David Maier	65	36,687
2	David J. DeWitt	89	38,770
3	Philip A. Bernstein	80	37,823
4	Michael Stonebraker	72	26,153
5	Michael J. Franklin	34	7,746
6	Serge Abiteboul	80	35,950
7	Jennifer Widom	101	63, 641
8	Joseph M. Hellerstein	90	43,207
9	Alon Y. Halevy	103	47,228
10	Jim Gray	81	46,884
11	Gerhard Weikum	88	34,028
12	Jeffrey F. Naughton	76	22,963
13	Yannis E. Ioannidis	59	15,017
14	Laura M. Haas	49	12,834
15	Stefano Ceri	77	29,506
16	Michael J. Carey	59	16,451
17	Rakesh Agrawal	108	124,595
18	Umeshwar Dayal	62	26,527

Table 8.1: Top communities by our approach in DS.

Chapter 9

Conclusion

In this study, we have introduced the keyword-aware influential community query (*KICQ*) that finds top- r most influential communities from an attributed graph, which has many practical applications. First, we have designed the *KICQ* as a set of query terms conjoining with predicates (AND or OR) that enables a user to search for influential communities from an attributed graph enriched by our proposed word-embedding based keyword similarity model. We have also proposed an influence measure for a community that considers both the cohesiveness and influence of individuals in the community. To answer the *KICQ* efficiently, we have developed two algorithms based on the derived upper bounds and results from already explored subgraphs. Our experimental results and case study show that our approach outperforms the state-of-the-art approach in both efficiency (up to 4 times faster) and effectiveness (up to 15 times higher keyword cohesiveness).

Bibliography

- [1] M. Sozio and A. Gionis, “The community-search problem and how to plan a successful cocktail party,” in *SIGKDD*, 2010, pp. 939–948.
- [2] J. Naruchitparames, M. H. Güneş, and S. J. Louis, “Friend recommendations in social networks using genetic algorithms and network topology,” in *2011 IEEE Congress of Evolutionary Computation (CEC)*. IEEE, 2011, pp. 2207–2214.
- [3] L. Kretz, “Virtual online community with geographically targeted advertising,” Apr. 17 2008, uS Patent App. 11/580,725.
- [4] Y. Fang, R. Cheng, S. Luo, and J. Hu, “Effective community search for large attributed graphs,” *PVLDB*, vol. 9, no. 12, pp. 1233–1244, 2016.
- [5] Y. Fang, R. Cheng, X. Li, S. Luo, and J. Hu, “Effective community search over large spatial graphs,” *PVLDB*, vol. 10, no. 6, pp. 709–720, 2017.
- [6] X. Huang and L. V. Lakshmanan, “Attribute-driven community search,” *PVLDB*, vol. 10, no. 9, pp. 949–960, 2017.
- [7] R.-H. Li, L. Qin, J. X. Yu, and R. Mao, “Influential community search in large networks,” *PVLDB*, vol. 8, no. 5, pp. 509–520, 2015.
- [8] J. Li, X. Wang, K. Deng, X. Yang, T. Sellis, and J. X. Yu, “Most influential community search over large social networks,” in *ICDE*, 2017, pp. 871–882.
- [9] Z. Zhang, X. Huang, J. Xu, B. Choi, and Z. Shang, “Keyword-centric community search,” in *ICDE*. IEEE, 2019, pp. 422–433.

-
- [10] L. Chen, C. Liu, K. Liao, J. Li, and R. Zhou, "Contextual community search over large social networks," in *ICDE*. IEEE, 2019, pp. 88–99.
- [11] D. Zhang, C.-Y. Chan, and K.-L. Tan, "Processing spatial keyword query as a top-k aggregation query," in *SIGIR*, 2014, pp. 355–364.
- [12] J. Li, C. Liu, and M. S. Islam, "Keyword-based correlated network computation over large social media," in *ICDE*, 2014, pp. 268–279.
- [13] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, "Keyword searching and browsing in databases using banks," in *ICDE*, 2002, pp. 431–440.
- [14] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar, "Bidirectional expansion for keyword search on graph databases," in *VLDB*, 2005, pp. 505–516.
- [15] H. He, H. Wang, J. Yang, and P. S. Yu, "Blinks: ranked keyword searches on graphs," in *SIGMOD*, 2007, pp. 305–316.
- [16] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu, "Finding maximal cliques in massive networks," *TODS*, vol. 36, no. 4, p. 21, 2011.
- [17] J. Cheng, Y. Ke, S. Chu, and M. T. Özsu, "Efficient core decomposition in massive networks," in *ICDE*, 2011, pp. 51–62.
- [18] J. Wang and J. Cheng, "Truss decomposition in massive networks," *PVLDB*, vol. 5, no. 9, pp. 812–823, 2012.
- [19] J. Moody and D. R. White, "Structural cohesion and embeddedness: A hierarchical concept of social groups," *American Sociological Review*, pp. 103–127, 2003.
- [20] S. Fortunato, "Community detection in graphs," *Physics reports*, vol. 486, no. 3, pp. 75–174, 2010.
- [21] Y. Zhou, H. Cheng, and J. X. Yu, "Graph clustering based on structural/attribute similarities," *PVLDB*, vol. 2, no. 1, pp. 718–729, 2009.

-
- [22] Y. Ruan, D. Fuhry, and S. Parthasarathy, “Efficient community detection in large networks using content and links,” in *WWW*, 2013, pp. 1089–1098.
- [23] Y. Jiang, C. Jia, and J. Yu, “An efficient community detection method based on rank centrality,” *Physica A: statistical mechanics and its applications*, vol. 392, no. 9, pp. 2182–2194, 2013.
- [24] Z. Xu, Y. Ke, Y. Wang, H. Cheng, and J. Cheng, “A model-based approach to attributed graph clustering,” in *SIGMOD*. ACM, 2012, pp. 505–516.
- [25] Y. Liu, A. Niculescu-Mizil, and W. Gryc, “Topic-link lda: joint models of topic and author community,” in *ICML*, 2009, pp. 665–672.
- [26] M. Sachan, D. Contractor, T. A. Faruque, and L. V. Subramaniam, “Using content and interactions for discovering communities in social networks,” in *WWW*. ACM, 2012, pp. 331–340.
- [27] J. Yang and J. Leskovec, “Overlapping community detection at scale: a nonnegative matrix factorization approach,” in *WSDM*. ACM, 2013, pp. 587–596.
- [28] J. Yang, J. McAuley, and J. Leskovec, “Community detection in networks with node attributes,” in *ICDM*. IEEE, 2013, pp. 1151–1156.
- [29] H. Wi, S. Oh, J. Mun, and M. Jung, “A team formation model based on knowledge and collaboration,” *ESA*, vol. 36, no. 5, pp. 9121–9134, 2009.
- [30] T. Lappas, K. Liu, and E. Terzi, “Finding a team of experts in social networks,” in *SIGKDD*, 2009.
- [31] C.-T. Li and M.-K. Shan, “Team formation for generalized tasks in expertise social networks,” in *SocialCom*, 2010, pp. 9–16.
- [32] A. Anagnostopoulos, L. Becchetti, C. Castillo, A. Gionis, and S. Leonardi, “Online team formation in social networks,” in *WWW*, 2012, pp. 839–848.

- [33] M. Kargar and A. An, “Discovering top-k teams of experts with/without a leader in social networks,” in *CIKM*, 2011, pp. 985–994.
- [34] D.-N. Yang, C.-Y. Shen, W.-C. Lee, and M.-S. Chen, “On socio-spatial group query for location-based social networks,” in *SIGKDD*, 2012, pp. 949–957.
- [35] W. Cui, Y. Xiao, H. Wang, and W. Wang, “Local search of communities in large graphs,” in *SIGMOD*, 2014, pp. 991–1002.
- [36] 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.
- [37] N. Barbieri, F. Bonchi, E. Galimberti, and F. Gullo, “Efficient and effective community search,” *Data mining and knowledge discovery*, vol. 29, no. 5, pp. 1406–1433, 2015.
- [38] D.-N. Yang, Y.-L. Chen, W.-C. Lee, and M.-S. Chen, “On social-temporal group query with acquaintance constraint,” *PVLDB*, vol. 4, no. 6, pp. 397–408, 2011.
- [39] L. Yuan, L. Qin, W. Zhang, L. Chang, and J. Yang, “Index-based densest clique percolation community search in networks,” *IEEE TKDE*, vol. 30, no. 5, pp. 922–935, 2017.
- [40] R.-H. Li, J. Su, L. Qin, J. X. Yu, and Q. Dai, “Persistent community search in temporal networks,” in *ICDE*. IEEE, 2018, pp. 797–808.
- [41] S. Chen, R. Wei, D. Popova, and A. Thomo, “Efficient computation of importance based communities in web-scale networks using a single machine,” in *CIKM*. ACM, 2016, pp. 1553–1562.
- [42] F. Bi, L. Chang, X. Lin, and W. Zhang, “An optimal and progressive approach to online search of top-k influential communities,” *PVLDB*, vol. 11, no. 9, pp. 1056–1068, 2018.
- [43] D. Zheng, J. Liu, R.-H. Li, C. Aslay, Y.-C. Chen, and X. Huang, “Querying intimate-core groups in weighted graphs,” in *2017 IEEE 11th International Conference on Semantic Computing (ICSC)*. IEEE, 2017, pp. 156–163.

- [44] S. Chobe and J. Zhan, “Advancing community detection using keyword attribute search,” *Journal of Big Data*, vol. 6, no. 1, p. 83, 2019.
- [45] A. Khan, L. Golab, M. Kargar, J. Szlichta, and M. Zihayat, “Compact group discovery in attributed graphs and social networks,” *Information Processing & Management*, p. 102054, 2019.
- [46] Y. Chen, Y. Fang, R. Cheng, Y. Li, X. Chen, and J. Zhang, “Exploring communities in large profiled graphs,” *IEEE TKDE*, 2018.
- [47] K. Wang, X. Cao, X. Lin, W. Zhang, and L. Qin, “Efficient computing of radius-bounded k-cores,” in *ICDE*. IEEE, 2018, pp. 233–244.
- [48] Q. Zhu, H. Hu, C. Xu, J. Xu, and W.-C. Lee, “Geo-social group queries with minimum acquaintance constraints,” *The VLDB Journal*, vol. 26, no. 5, pp. 709–727, 2017.
- [49] R.-H. Li, L. Qin, F. Ye, J. X. Yu, X. Xiao, N. Xiao, and Z. Zheng, “Skyline community search in multi-valued networks,” in *SIGMOD*. ACM, 2018, pp. 457–472.
- [50] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS’13. USA: Curran Associates Inc., 2013, pp. 3111–3119. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2999792.2999959>
- [51] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *arXiv preprint arXiv:1301.3781*, 2013.
- [52] Y.-B. Kang, P. Delir Haghighi, and F. Burstein, “Cfinder: An intelligent key concept finder from text for ontology development,” *Expert Syst. Appl.*, vol. 41, no. 9, pp. 4494–4504, Jul. 2014. [Online]. Available: <http://dx.doi.org/10.1016/j.eswa.2014.01.006>
- [53] Y.-B. Kang, P. D. Haghighi, and F. Burstein, “Taxofinder: a graph-based approach for taxonomy learning,” *IEEE TKDE*, vol. 28, no. 2, pp. 524–536, 2015.

- [54] V. Batagelj and M. Zaversnik, “An $O(m)$ algorithm for cores decomposition of networks,” *arXiv preprint cs/0310049*, 2003.
- [55] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, “Arnetminer: extraction and mining of academic social networks,” in *SIGKDD*, 2008, pp. 990–998.
- [56] N. Seco, T. Veale, and J. Hayes, “An intrinsic information content metric for semantic similarity in wordnet,” in *Ecai*, vol. 16, 2004, p. 1089.
- [57] Y. Wang, L. Wang, Y. Li, D. He, and T.-Y. Liu, “A theoretical analysis of ndcg type ranking measures,” in *Conference on Learning Theory*, 2013, pp. 25–54.
- [58] J. J. Jiang and D. W. Conrath, “Semantic similarity based on corpus statistics and lexical taxonomy,” *arXiv preprint cmp-lg/9709008*, 1997.
- [59] D. L. Davies and D. W. Bouldin, “A cluster separation measure,” *IEEE transactions on pattern analysis and machine intelligence*, no. 2, pp. 224–227, 1979.
- [60] 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*, Jul 2019. [Online]. Available: <https://doi.org/10.1007/s00778-019-00556-x>