




Robust keyword search in large attributed graphs

Spencer Bryson¹ · Heidar Davoudi¹ · Lukasz Golab² · Mehdi Kargar³  ·
Yuliya Lytvyn² · Piotr Mierzejewski⁴ · Jaroslaw Szlichta^{1,4} · Morteza Zihayat^{3,4}

Received: 4 November 2019 / Accepted: 13 July 2020 / Published online: 22 July 2020
© Springer Nature B.V. 2020

Abstract

There is a growing need to explore attributed graphs such as social networks, expert networks, and biological networks. A well-known mechanism for non-technical users to explore such graphs is keyword search, which receives a set of query keywords and returns a connected subgraph that contains the keywords. However, existing approaches, such as methods based on shortest paths between nodes containing the query keywords, may generate weakly-connected answers as they ignore the structure of the whole graph. To address this issue, we formulate and solve the *robust keyword search* problem for attributed graphs to find strongly-connected answers. We prove that the problem is NP-hard and we propose a solution based on a random walk with restart (RWR). To improve the efficiency and scalability of RWR, we use Monte Carlo approximation and we also propose a distributed version, which we implement in Apache Spark. Finally, we provide experimental evidence of the efficiency and effectiveness of our approach on real-world graphs.

Keywords Attributed graphs · Social networks · Keyword search

1 Introduction

Increasingly, organizations and communities are focusing on graph analysis to make faster and better decisions leading to business and societal impact. Attributed graphs, i.e., those with attributes or labels attached to their nodes and/or edges, are especially common. Examples include social networks (e.g., Facebook and Twitter, whose nodes may be labelled with user information) and biological networks (e.g., protein-protein interaction networks whose nodes are labeled with protein properties). Moreover, other data models can also be naturally represented as attributed graphs. For example, a relational database instance corresponds to a graph whose nodes are tuples (labelled with the attributes of the

✉ Mehdi Kargar
kargar@ryerson.ca

✉ Morteza Zihayat
mzihayat@ryerson.ca

Extended author information available on the last page of the article

Customer

C_ID	C_Name	Email
10	David Jackson	d@gm.ca
20	Sarah Jackson	s@ce.us

Customer_Account

CA_ID	CA_C_ID	CA_Name
22	10	Saving Account

Trade

T_ID	Date	Price	Quantity	T_CA_ID
52	2019-05-05	\$40.25	100	22

Security

S_SYMB	S_CO_ID	S_Name
ADTK	84	COMMON of ADTK

Company

CO_ID	CO_Name
84	Adept Inc.

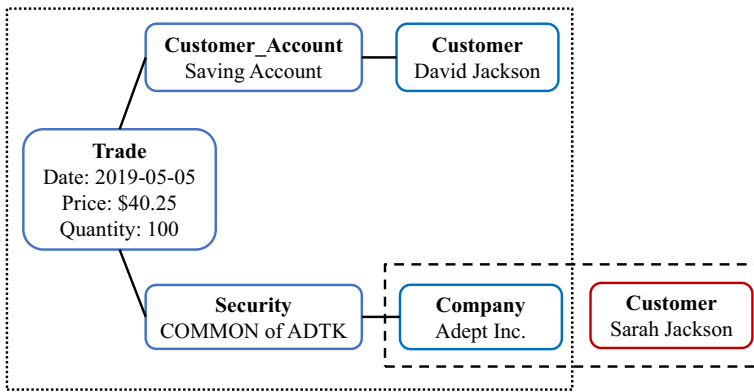


Fig. 1 A portion of the TPC-E database and its associated attributed graph

tuples) and whose edges denote foreign key relationships. Given the wealth of information contained in attributed graphs, exploring and mining them is of critical importance (Kargar et al. 2014; Wang et al. 2010; Markowetz et al. 2007).

Over the last decade, keyword search over graphs has been used to explore and analyze graph databases. An important advantage of keyword search over other graph analytics platforms is that it does not require the knowledge of the graph database schema or a query language such as SQL or SPARQL (Prud’hommeaux 2008). Given an attributed graph and a query composed of a set of keywords, the answer is a connected subgraph that contains all the query keywords. To motivate the need to return a connected subgraph, we present a keyword search example over the TPC-E benchmark database.

Figure 1 shows a fragment of the TPC-E¹ benchmark database, which models and stores on-line transaction processing (OLTP) data of a brokerage firm. The database contains financial information such as broker fees, traded companies, and customer accounts. By converting foreign keys into edges, the associated graph is shown below the tables in Fig. 1. Consider the query “Jackson Adept” over the TPC-E database. The goal is to find

¹ <http://www.tpc.org/tpce/>.

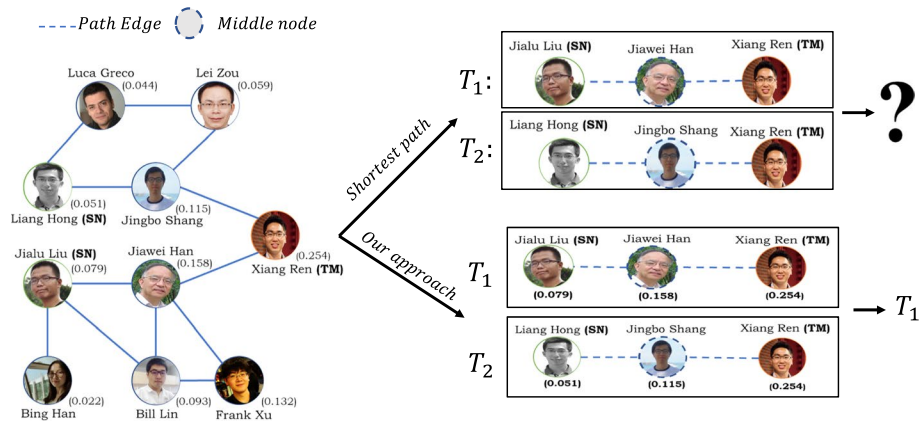


Fig. 2 A portion of the DBLP expert network. The numbers on each node are the RWR scores when the restart node is Xiang Ren

the relationship between a customer “Jackson” and a company “Adept Inc.”. If we take into account the connections among nodes (i.e., edges of the graph), a possible answer is the subgraph that connects customer “David Jackson” and “Adept Inc.”. This subgraph is shown inside the dotted box in Fig. 1, and indicates that the stock of Adept Inc. was traded by “David Jackson” in May 2019. This is an interesting and strong relationship between a company and a customer. On the other hand, if connections between nodes are ignored, and the only goal is to return a set of nodes that cover the input keywords, customer “Sarah Jackson” and company “Adept Inc.” is a possible answer. This answer is shown in the dashed box in Fig. 1. However, this disconnected subgraph is not an informative answer as it does not reveal any relationship between a customer and a company. Without the connectivity constraint, this problem becomes the classical set cover problem.

A technical challenge in this context is to rank the qualifying subgraphs and select the best one. To do so, recent work (e.g., Yang et al. 2019; Kargar et al. 2016) uses the notion of shortest paths. For example, potential answers may be scored based on the diameter of the subgraph or the sum of the shortest paths between all pairs of nodes containing the query keywords. However, the shortest path approach lacks robustness since it may not reflect the overall structure of the answer subgraph. In this work, we propose an alternative approach to measure the distance between nodes containing the query keywords, with the goal of capturing the graph structure and thus improving the quality of the answers. To motivate our approach, we illustrate the shortcomings of existing work based on shortest paths in two different applications:

Application 1: Academic collaborations Figure 2 illustrates a fragment of the DBLP co-authorship network,² whose nodes correspond to authors and node attributes correspond to the expertise of the authors (similar to Lappas et al. 2009, the expertise of each author consists of terms that appear in at least two titles of papers that they co-authored). Edges represent collaborations, i.e., co-authoring a paper. Suppose we are seeking experts in *Social Networks (SN)* and *Text Mining (TM)* for a project. To do so, we issue the corresponding keyword query to find a subgraph of the DBLP network containing experts in these two

² <http://www.informatik.uni-trier.de/~ley/db/>.

areas. As illustrated in Fig. 2, there are two teams (e.g., subgraphs) that cover the required skills: $T_1 : \{Jialu\ Liu, Jiawei\ Han, Xiang\ Ren\}$ and $T_2 : \{Liang\ Hong, Jingbo\ Shang, Xiang\ Ren\}$. Both T_1 and T_2 cover the two query keywords, (SN) and (TM), and may include additional nodes, called *middle nodes*, that connect the nodes containing the query keywords. The shortest path from *Liu* to *Ren* passes through *Jiawei Han*, the shortest path from *Hong* to *Ren* goes through *Jingbo Shang*, and both have length 2. Thus, a shortest path approach gives the same score to these two teams. However, *Ren* is connected to *Liu* through multiple paths (e.g., via *Jiawei Han* who is a senior researcher and himself has many connections to other researchers). Hence, we argue that *Ren* forms a more robust team with *Liu* than with *Hong*. Moreover, *Liu* has more connections than *Hong*. Our solution takes these properties, and the overall structure of the answer subgraph, into account when scoring the answers. As we will describe later, we do so by measuring the proximity between nodes containing the query keywords using a random walk rather than the length of the shortest paths. In particular, random walk assigns weights to each node during our answer ranking process, as shown in Fig. 2. We select T_1 over T_2 because the assigned node weights are higher. Note that this application of keyword search in attributed graphs corresponds to the team formation problem introduced in Lappas et al. (2009). A solution for one can be used to solve the other one and vice versa (see the related work in the next section for details).

Application 2: Protein–protein interaction networks Proteins are important parts of every cell and their interactions affect various biological mechanisms (Gonzalez and Kann 2012). In a protein-protein interaction network (PPI), nodes are proteins and an edge between two proteins denotes their interaction. Furthermore, each protein has a level of expression: the higher the level, the higher the chance of formation (Duret and Mouchiroud 1999). Thus, proteins with higher expression levels are candidates for drug discovery and enzyme analysis (Duret and Mouchiroud 1999). The length and structure of a protein is related to its expression level (Duret and Mouchiroud 1999; Ingvarsson 2007a). In this example, we use IntAct PPI³ to build a PPI and calculate each protein’s expression level, which becomes the corresponding node weight. Each protein is encoded by a set of different genes. Thus, genes are attributes (i.e., keywords) of proteins. It is known that some diseases are controlled by some genes (Pinero 2017). In order to control a disease that is caused by a set of particular genes, we aim to find a set of proteins in the PPI graph that are encoded by these genes and form a subgraph. From the drug discovery perspective, we want to study the effect of silencing these proteins on the advancement of a disease (Jay 2019). Thus, genes are the query keywords in our framework, and the output is a subgraph of interacting proteins from a PPI network.

Figure 3 shows a portion of a PPI network. Suppose we want to find a subgraph of this PPI network that encodes two genes: *N* and *HLAB*. Two possible answers in Fig. 3 are: A_1 , which includes $\{P30480, Q8JPQ9\}$, and A_2 , which includes $\{P30480, P03418\}$; here, protein *P30480* encodes gene *HLAB*, and proteins *Q8JPQ9* and *P03418* encode gene *N*. In terms of the shortest path distance, both A_1 and A_2 have the same cost—of two—which is the shortest path between the nodes holding the keywords *N* and *HLAB*.⁴ However, answer A_1 has stronger connectivity: there are two shortest paths of length two between its content nodes, versus only one in A_2 . We also show the cost of each

³ <https://www.ebi.ac.uk/intact>.

⁴ We call these nodes (that contain the query keywords) content nodes or keyword holders, in contrast to middle nodes that serve as connections between the content nodes.

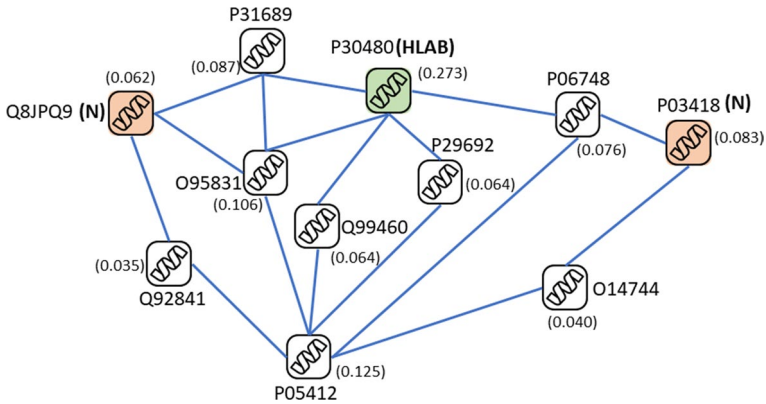


Fig. 3 A portion of a PPI network to answer query $\{N, HLAB\}$. The expression levels of proteins are shown as node weights in brackets. Note that lower node weights correspond to higher expression levels

protein in Fig. 3 as node weights. The lower the cost, the higher the expression level (and higher expression is preferred). The cost of protein Q8JPO9 in A_1 is lower than the cost of protein P03418 in A_2 . Thus, A_1 has a higher expression level and therefore is more desirable for biological studies. However, without taking the entire graph structure into account, A_1 would not be chosen over A_2 .

Motivated by the above examples, we propose a new method, based on a random walk with restart, to rank the results (subgraphs) of keyword queries over attributed graphs. Our method takes the structure of the entire graph into account to identify robust answers. Using real-world graphs, we experimentally show that our approach produces higher-quality results than the previous state of the art. Our contributions are as follows:

1. We define the problem of *robust keyword search in large attributed graphs* and show that considering the entire graph structure during the search process results in higher-quality answers. Our approach assigns specific query-dependent scores to nodes, in contrast to prior graph mining work that computes PageRank scores corresponding to the overall importance of nodes.
2. We prove that the problem is NP-hard and propose an approach to find answers using a random walk with restart (RWR).
3. We propose a distributed framework to discover answers from large networks using RWR by developing efficient partitioning and load-balancing techniques. The framework is general enough that it can support any RWR algorithm (e.g., the power iteration method and the approximate Monte Carlo method).
4. We provide extensive experiments on real-world graph datasets (i.e., DBLP and PPI networks) to demonstrate that our approach scales well to large graphs and returns more relevant answers than previous approaches.

The remainder of this paper is organized as follows. Related work is discussed in the next section. We discuss preliminaries and the problem statement in Sect. 3. We present the RWR method to solve the problem of robust keyword search over graphs in Sect. 4. Experiments are presented in Sects. 5 and 6 concludes the paper.

2 Related work

2.1 Keyword search in graphs

Graph keyword search takes a set of keywords as input and returns a subgraph that contains the input keywords. Connectivity of the answer subgraph is of critical importance as it shows the underlying relation among the keyword holders. A number of works addressed this problem in the past and they can be categorized based on the types of answers they produce: (1) tree-based and (2) graph-based. Tree-based methods return two types of trees: Steiner trees and distinct root trees. A dynamic programming algorithm for finding Steiner trees was introduced in Ding et al. (2007). It has exponential runtime complexity in terms of the the number of input keywords. However, it is polynomial in terms of the number of nodes in the graph. A bidirectional search algorithm was proposed in Kacholia et al. (2005) and improved in BLINKS (He et al. 2007) for finding trees with distinct roots. Part of the scoring function that was proposed by BLINKS assigns PageRank scores to content nodes (called matches in BLINKS). However, unlike our work, the PageRank scores in BLINKS are static and independent of the query keywords. We capture the relative importance of nodes with respect to other nodes, depending on the query keywords, using the RWR process.

For graph based methods, the work of Li et al. (2008) produces r -radius Steiner graphs and the work of Qin et al. (2009) returns multi-centered subgraphs. Each answer subgraph is built around a set of center nodes. In Qin et al. (2009), two algorithms are proposed to find *communities* in a graph: 1) an algorithm that generates all communities in arbitrary order, and 2) an algorithm that generates top- k communities. In Kargar and An (2011), the authors produce r -cliques as answers to keyword search over graphs. An r -clique is a subgraph in which all content nodes are guaranteed to be close to each other. Generating duplication-free and minimal answers for graph keyword search is proposed in Kargar et al. (2014). Authors of Kargar et al. (2016) propose a greedy algorithm to search over heterogeneous graphs that took node costs into account. The authors of Termehchy and Winslett (2011) propose to rank answers to keyword search over XML documents based on the structural information of sub-trees and the degree of relatedness between keywords and sub-trees.

Random walk and PageRank have been used to rank Web pages and in other search problems. ObjectRank ranks answers based on the importance and authority of keyword holders by extending the PageRank method (Balmin et al. 2004). Unlike our work, the output of ObjectRank is a ranked list of nodes (our output is a ranked list of subgraphs). A more efficient version of ObjectRank was proposed by Chakrabarti (2007). The authors of Lao and Cohen (2010) solve the problem of relational retrieval using an efficient algorithm that calculates path-constrained random walks. The input to this search problem is a weighted set of graph nodes (unlike our problem, in which the input consists of a set of query keywords), and the output is a ranked set of answer nodes that are ordered by proximity to the query nodes.

Keyword search has been used to explore Resource Description Framework (RDF) data. The work on keyword search over RDF data can be divided into three categories. In the first category, the RDF data is converted into a graph, on which graph keyword search techniques are directly applied (Kasneji et al. 2009). The second approach is based on summarization (Le et al. 2014). In the first step, the hierarchy of RDF class nodes is employed to produce a summary graph. In the next step, a graph algorithm is used on the

summary graph to generate relevant answers. The third category has recently been proposed in Han et al. (2017). In the first step, and based on the input keywords, the algorithm finds elementary query graph building blocks. In the next step, keyword search over RDF data is formulated as a query graph assembly problem. However, none of these approaches consider robust measures that take into account the entire graph structure when finding relevant answers.

2.2 Team formation in social networks

The problem of keyword search in attributed graphs is related to the problem of team formation in expert networks, introduced by Lappas et al. (2009). Each expert possesses a set of skills (e.g., Java or SQL), and experts are connected to each other based on their past experience (e.g., working on the same project). Given a network of experts, and a set of required skills to complete a project, the goal is to find a subgraph of this network in which members of the subgraph (i.e., team) collectively cover all the required skills. Qualifying subgraphs are ranked according to some objective function that favours connectedness and minimizes communication cost. One objective function discussed by Lappas et al., was the diameter of the subgraph (Lappas et al. 2009). Recently, the authors of Yin et al. (2019) proposed a new objective function inspired by the objective function of classical Group Steiner Tree problems. They also proposed a greedy algorithm to optimize this objective.

The original team formation problem is similar to the graph keyword search problem, meaning that our solution to the latter problem is directly applicable to the former problem (as we discuss in our first motivating example in the Introduction). Due to the nature of expert networks and special circumstances, a variety of methods were proposed over the last decade to address different requirements.

Given a capacity for each expert, the authors of Majumder et al. (2012) propose a method that guarantees no user is assigned a task beyond their capacity. As stated by the authors, their work is different from the original team formation problem as Lappas et al. do not consider the capacity constraints of experts (Majumder et al. 2012). The authors of Kargar et al. (2013) find a team of experts that minimizes the communication cost as well as the personnel cost. The idea is that each expert requests a salary to participate in a project. Since each project has a limited budget, we do not want to hire experts beyond the project's budget. Adding the personnel cost makes this a bi-objective optimization problem that is different than the original team formation problem.

The authors of Wang et al. (2019) study the problem of forming a truthful team of experts. In addition of forming teams with maximal social welfare, the proposed mechanism in Wang et al. (2019) ensures that all team members truthfully report their participation and skills. The problem of employee training was introduced in Zhang et al. (2017). In this problem, employees are assigned to various projects. As stated by the authors of Zhang et al. (2017), this problem is different from the original team formation problem as it partitions employees into different teams for several projects. All employees (i.e., experts) will be assigned to at least one project. On the other hand, in the original team formation problem by Lappas et al., one team (with few experts) is selected to cover only one project.

Recently, the authors of Jeong et al. (2019) proposed a distributed framework over Apache Spark to find teams of experts. The proposed method has three main phases. In the first phase, the graph is distributed among nodes in a cluster. In each node of the cluster, the experts who have at least one of the required skills are found. The results are represented as subgraphs. In the second phase, the subgraphs are merged to form potential teams. In the last phase, each

merged graph is evaluated if it contains at least one required skill. This approach is different from our distributed method. First, the graph partitioning in Jeong et al. (2019) is random while we use custom graph partitioning. Second, the work in Jeong et al. (2019) produces many subgraphs in each worker, many of which may not cover all the required skills. This may severely affect the performance of the algorithm. In contrast, our approach starts from the rarest-skill holders and therefore the number of generated teams is significantly lower than in Jeong et al. (2019). Third, we optimize the RWR based proximity function while the work in Jeong et al. (2019) optimizes the shortest path based method proposed by Lappas et al. (2009).

In addition to optimizing the communication cost, recent work on team formation adds new objectives. In this work, we focus on the keyword search problem (and the original team formation problem) and propose a new metric (that is fundamentally different than the shortest path-based methods) to define the proximity between nodes in the network (based on the concept of random walk with restart). In future work, we plan to incorporate these additional requirements into our proximity measure.

2.3 Community search in attributed graphs

A community is defined as a densely connected subgraph in an input graph (Bai et al. 2018) and may contain thousands of nodes (Hajiabadi et al. 2017). The problem of community detection is to find all such subgraphs. Recently, the problem of community search has been proposed. The aim of the community search is to find *specific* communities that are formed around a specific *query node* (Li et al. 2015). Therefore, unlike classical community detection, the problem of community search receives an input node and builds communities around the input node (also called query node). A variety of density-based metrics (e.g., minimum degree) is used to determine the density of communities. For example, the works in Sozio and Gionis (2010); Cui et al. (2014) return k -cores around the query node (in a k -core subgraph, each node has a degree of at least k).

Recently, Fang et al. (2016) proposed the problem of community search in attributed graphs. The input is a set of query keywords, a query node, and the minimum degree of nodes in a community. The result is a set of communities (i.e., k -cores) around the query node, in which the minimum degree requirements are satisfied and nodes contain the input keywords. Following the work of Fang et al. (2016), Huang and Lakshmanan (2017) proposed another community search algorithm over attributed graphs that receives a group of query nodes instead of a single node. Also, in Huang and Lakshmanan (2017), the returned community is a k -truss. In a k -truss subgraph, each edge is part of at least $(k - 2)$ triangles. The problem of keyword search in attributed graphs is different than community search in attributed graphs. First, we search the entire graph, while community search receives a query node (or a set of query nodes) and builds the community around that node(s). Second, the size of our returned subgraph is small, while a community might have thousands of nodes. Note that in our work, the number of content nodes is at most equal to the number of input keywords. Third, in our work, usually one node covers one query keyword. However, in community search, many nodes usually contain each query keyword.

3 Preliminaries and problem statement

Let $G = (V, E, K)$ be an attributed graph,⁵ where the set of n nodes (i.e., vertices) is specified by $V = \{v_1, v_2, \dots, v_n\}$, the set of p edges is specified by $E = \{e_1, e_2, \dots, e_p\}$ and the set of m keywords (i.e., node labels) is specified as $K = \{k_1, k_2, \dots, k_m\}$. For simplicity of presentation, we assume G is unweighted and undirected, but our proposed algorithms can handle weighted and directed graphs as well. Each node v_i is associated with a set of keywords $K(v_i) \subseteq K$. A subset of nodes $V' \subseteq V$ contains keyword k_j if at least one of the nodes in V' contains k_j . For each keyword k_j , the set of all nodes that contain k_j is denoted as $V(k_j) = \{v_i | k_j \in K(v_i)\}$. A query Q is a set of keywords: $Q \subset K$. Finally, a subset of nodes $V' \subseteq V$ covers a query Q if $\forall k_j \in Q, \exists v_i \in V' : k_j \in K(v_i)$.

Definition 1 (*Candidate Answer*) Given a graph G and a query Q containing a set of q keywords $Q = \{k_1, k_2, \dots, k_q\}$, a *candidate answer* C is a subgraph of G whose nodes cover Q .

In other words, given a query Q , a candidate answer C is represented by a set of q keyword-node pairs: $C = \{v_{k_1}, v_{k_2}, \dots, v_{k_q}\}$, where v_{k_j} is a node in C that contains keyword k_j . Note that the same node may contain multiple keywords, i.e., the v_{k_j} s are not necessarily distinct.

A candidate answer contains all the query keywords, but some candidates may be more closely connected than others. To rank the candidate answers, we define a *connection score function* $sc(v_i, v_j)$ that assigns a score to each pair of nodes v_i and v_j . The higher the score $sc(v_i, v_j)$, the stronger the relationship between nodes v_i and v_j in graph G . Existing approaches define this function as the shortest path between v_i and v_j in G (Anagnostopoulos et al. 2012; Lappas et al. 2009; Kargar et al. 2013). In other words, the smaller the shortest distance between v_i and v_j , the higher the score $sc(v_i, v_j)$. In this work, we design a score function that takes the entire graph structure into account (to be discussed in next sections). For now, assuming that such a function exist, we define the connection score of a candidate answer C as follows.

Definition 2 (*Connection Score of a Candidate Answer*) Given a candidate answer C that covers query $Q = \{k_1, k_2, \dots, k_q\}$, the connection score of C is defined as:

$$ConSc(C) = \sum_{i=1}^q \sum_{j=i+1}^q sc(v_{k_i}, v_{k_j}) \quad (1)$$

The connection score sums up the scores between all pairs of *content nodes*. Thus, each keyword and each content node contributes equally to the connection score function. On the other hand, other commonly used functions (such as the diameter of the answer subgraph) might be biased towards a small fraction of the content nodes (only two nodes contribute to the calculation of the diameter).

⁵ We assume G is connected. If G is not connected, we use the strongly connected component, as in Lappas et al. (2009). Having a connected graph guarantees that the returned answers are always connected. Note that in our applications, returning disconnected subgraphs is meaningless.

Recall that the content nodes v_{k_i} need not be unique since the same node may contain multiple query keywords. Thus, the number of content nodes in an answer can range from one to q , where q is the number of query keywords. While maximizing *ConSc* does not explicitly minimize the number of content nodes, it does ensure that the selected content nodes are robustly connected—a requirement illustrated by our motivating examples. Furthermore, maximizing *ConSc* does not imply the need to use q unique content nodes. If the same node is a content node for w keywords, its $sc()$ scores will appear w times in the summation in Eq. 1.

Below, we define the problem of robust keyword search over attributed graphs.

Problem 1 (Robust Keyword Search over a Graph) Given a graph G , a query Q and the score function sc , find a candidate answer C for Q with a maximal connection score $ConSc(C)$.

Theorem 1 *Problem 1 is NP-hard.*

Proof We prove that the decision version of Problem 1 is NP-hard. Therefore, and as a direct result, Problem 1 is also NP-hard. The decision version asks whether there exists a candidate answer with the minimum connection score of b for some constant b . Clearly, the problem is in NP. We prove this theorem by a reduction from 3-SAT. Consider a set of m clauses $F_k = x_k \vee y_k \vee z_k$ ($k = 1, \dots, m$) and $\{x_k, y_k, z_k\} \subset \{u_1, \bar{u}_1, \dots, u_n, \bar{u}_n\}$. The score between each variable and its negation (i.e. u_i and \bar{u}_i) is set to ϵ where $\epsilon = \frac{b}{\binom{n+m}{2}^2}$. The score between other variables is set to $\frac{b+1}{\binom{n+m}{2}}$. An instance of Problem 1 is defined next. Two nodes are created for each pair of variables u_i and \bar{u}_i . Thus, there exist $2 \times n$ nodes. For each pair of variables u_i and \bar{u}_i , one keyword KW_i ($i = 1, \dots, n$) is generated. Therefore, u_i and \bar{u}_i contain keyword KW_i . Note that u_i and \bar{u}_i are the only holders of KW_i . Furthermore, for each clause F_k , one keyword KW_{n+k} ($k = 1, \dots, m$) is generated. The holders of KW_{n+k} are the triples of nodes that are related to x_k, y_k and z_k . Thus, the total number of required keywords is $n + m$. One feasible solution to the problem we described with the minimum connection score of b is as follows. From each pair of nodes associated to u_i and \bar{u}_i , one of them is chosen. Also, from each triple of nodes associated to x_k, y_k and z_k , one is chosen. Therefore, the existence of a subset of nodes with the minimum connection score of b is equivalent to the existence of a satisfying assignment for $F_1 \wedge F_2 \wedge \dots \wedge F_m$. Also, a satisfying assignment guarantees the existence of a set of nodes with the minimum connection score of b . This completes the proof. \square

As we proved in Theorem 1, regardless of how we define the score function (i.e., sc), Problem 1 is NP-hard. Various methods for the problem of keyword search over graphs have been studied in the past (see, for example, Bhalotia et al. 2002; Ding et al. 2007; Kargar and An 2011; Kargar et al. 2014; Lappas et al. 2009). However, to the best of our knowledge, existing approaches do not use a robust scoring function that considers the entire graph structure. As a result, the connection score is also limited to a small portion of the graph. In contrast, we leverage the structural properties of the entire graph in defining a score function (and the connection score), and we design an efficient algorithm to discover the best candidate answer. While using robust score functions is desirable (recall our motivating examples), it introduces new technical challenges. In the next section, we propose a new scoring measure and we present our solution to address the complexity issues in employing this measure in keyword search over graphs.

4 Robust keyword search using random walk with restart (RWR)

Random walk with restart is a robust method to assign a connection score to each node relative to other nodes (i.e., *resSet*) in the entire graph. A random walker (i.e., surfer) starts from a node in *resSet* and “surfs” the nodes by randomly choosing which edges to follow. Moreover, the random walker might teleport to a random node in *resSet* with probability β . When the process reaches a stationary condition, the score vector \mathbf{p} is as follows:

$$\mathbf{p} = (1 - \beta) * W\mathbf{p} + \beta * \mathbf{r}, \quad (2)$$

where $\mathbf{p} = [p_j]$, p_j is a relevance score of node v_j w.r.t. the node(s) in *resSet*, $W = [w_{ij}]$ is the transition probability matrix (w_{ij} is the probability of transitioning from node v_i to node v_j), and $\mathbf{r} = [r_i]$ is the starting vector, where $r_i = 1$ if $i \in \text{resSet}$ and 0 otherwise. We define the RWR-based score function between two nodes v_i and v_j as follows:

$$sc_{RWR}(v_i, v_j) = p_j, \text{ where } r_j = 1 (r_k = 0 \text{ for all } k \neq i) \quad (3)$$

Next, we present an algorithm that uses the RWR score function to solve Problem 1. As we discussed, Problem 1 is NP-hard. Therefore, we propose an efficient and effective heuristic to solve it in polynomial time.

The idea is as follows. For a given query Q , take each node containing the rarest keyword⁶ (i.e., each node in $V(k_{rare})$) and form a subgraph (candidate answer) around that node. The answer’s score is initialized to 0. Then, in each iteration, we cover one of the uncovered keywords. In order to do that, we run the RWR and set the restart nodes to the current nodes of the subgraph. At the beginning, the only node in the subgraph is the one with the rarest keyword. When the RWR process is finished, the node with the highest score that also covers the current required keyword is selected as the best node of the subgraph. Among all the candidate subgraphs that are formed around the node containing the rarest keywords, the one with the highest sum of RWR scores is selected as the best subgraph.

Algorithm 1, represents this process in detail. The input is the graph G , the query Q , and the keyword holders of (i.e., the nodes that contain) each keyword. The output is a subgraph (i.e., the most robust candidate answer) that covers the required keywords. The complexity of Algorithm 1 is $O(|V(k_{rare})| \times q \times RWR_{time})$, where $|V(k_{rare})|$ is the number of rarest keyword holders, q is the number of query keywords, and RWR_{time} is the runtime of RWR. Note that RWR_{time} is linear when using Monte Carlo approximation and is sub-linear in a distributed environment (as we will explain shortly).

To find the top- k best answers, we additionally initialize a list L of size k for the output. The list L is updated after each iteration of the for-loop and the new candidate answer is added to L if its connection score is lower than the connection score of the highest-score answer in L . The runtime complexity remains the same as we only need an extra pass over L in each iteration and the size of L is limited by k .

Recall the example in Fig. 2 in which we need to cover two keywords, Text Mining (TM) and Social Network (SN). Since TM is the rarest keyword as it is only held by one

⁶ Note that choosing any set of nodes that contain one of the input keywords in the first step guarantees the coverage of all keywords at the end. However, starting with the rarest keyword holder is a greedy strategy that improves performance significantly as it prunes the search space without significantly sacrificing precision in practice.

node (i.e., Ren), we form a subgraph around *Ren*. We run a RWR process, setting *Ren* as the restart node and setting β to 0.15. The score of each node after the RWR process is also shown in Fig. 2. Between the two nodes containing keyword *SN*, *Liu* is selected since his RWR score is higher than *Hong* (the other holder of *SN*). The robustness of our team is evident: shortest path approaches cannot distinguish between *Hong* and *Liu*; however, by evaluating the entire graph using RWR, *Liu* is considered to be a more effective collaborator to work with *Ren*.

Algorithm 1 Robust Keyword Search using Random Walk with Restart (RWR)

Input: graph G with n nodes; query $Q = \{k_1, k_2, \dots, k_q\}$; the set of nodes that contain each keyword k_i , $V(k_i)$, for $i = 1, \dots, q$.

Output: a robust subgraph that covers all input keywords

```

1:  $bestAnswer \leftarrow \emptyset$ 
2:  $maxScore \leftarrow -\infty$ 
3:  $k_{rare} \leftarrow$  keyword in  $Q$  held by the fewest nodes
4: for each  $v_r \in V(k_{rare})$  do
5:    $answer \leftarrow$  initialize a new candidate answer (i.e., subgraph)
6:    $score \leftarrow 0$ 
7:    $answer.add(\langle k_{rare}, v_r \rangle)$ 
8:   for  $j \leftarrow 1$  to  $q$  and  $k_j \neq k_{rare}$  do
9:      $resSet \leftarrow$  current members of  $answer$  subgraph
10:    Run a random walk with restart (RWR) to set  $resSet$ 
11:     $p_j \leftarrow \{p \in V(k_j) \mid p \text{ gets highest score in the RWR}\}$ 
12:     $answer.add(\langle k_j, p_j \rangle)$ 
13:     $score_j \leftarrow$  the score of  $p_j$  in RWR
14:     $score \leftarrow score + score_j$ 
15:   if  $score > maxScore$  then
16:      $bestAnswer \leftarrow answer$ 
      $maxScore \leftarrow score$ 
17: return  $bestAnswer$ 

```

4.1 Monte carlo approximation of RWR

The main computational bottleneck in Algorithm 1 is line 10, where RWR scores for the experts need to be calculated. RWR is traditionally computed via power iteration, which is prohibitively expensive for large graphs. However, there exists a Monte Carlo method that provides an accurate estimate to RWR with significantly lower runtime (Bahmani et al. 2010). The Monte Carlo method simulates “surfers” traversing the graph at random, and does not require the entire graph as a $n \times n$ matrix in each iteration, where n is the number of nodes.

To estimate RWR for a single node, we create N_{rw} random walkers as (*source*, *currentPosition*) tuples, where the *source* and *currentPosition* are initialized at the node in G that corresponds to the node(s) we are starting the walk from. The walkers follow the same parameters as the standard approach, i.e. each walker takes n steps (or iterations), and at each step the walker has a probability, β , to restart on the *source* node or a $(1 - \beta)$

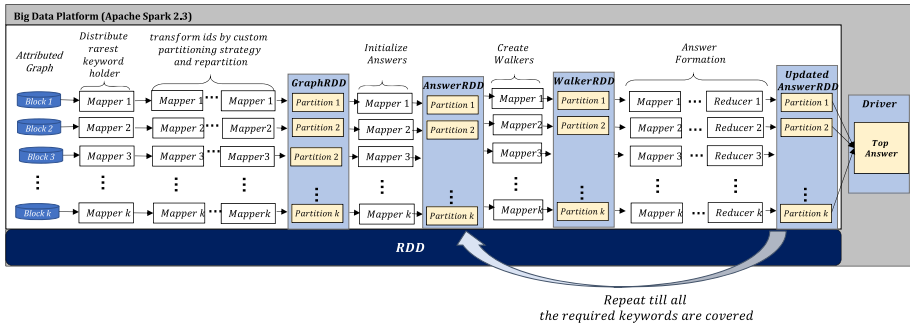


Fig. 4 The conceptual distributed framework for robust keyword search in large graphs

probability to continue walking, where it moves to a random node adjacent to *currentPosition*. The random walkers will continue to walk until termination when all steps are taken. The final distribution of walkers represent an accurate estimate of RWR for nodes. If a node is too far or receives no walkers, it is assigned a score of 0. As this is a Monte Carlo method, we can increase the accuracy of the approximation by increasing the number of random samples we perform. This is analogous to increasing the number of random walkers we start on each member of $V(k_{rare})$.

4.2 Distributed algorithm for RWR

In this section, we design a distributed version of Algorithm 1. Technically, each walker in our approach is independent and each walk can be parallelized. Figure 4 shows the overview of the proposed distributed framework, implemented in Apache Spark.

Resilient distributed datasets (RDD) variables First, we describe variables related to Resilient Distributed Datasets⁷ (RDD). We create three variables: *GraphRDD*, *WalkerRDD* and *AnswerRDD*. *GraphRDD* is the input graph, which is divided into into h random partitions. Later, *GraphRDD* holds the partitions created by our proposed custom partitioning strategy. *WalkerRDD* contains the walkers with the same number of partitions as *GraphRDD*. Since these were created using the same partitioner, the partitions within *WalkerRDD* and *GraphRDD* correspond one-to-one and as a result become co-located. This helps us obtain iterators over both the *WalkerRDD* and *GraphRDD* partitions. Thus, we can iterate over the *WalkerRDD* partition and select the neighbors of a walker from the *GraphRDD* partition iterator. The last variable is *AnswerRDD*, which keeps track of candidate answer subgraphs found by the algorithm.

Custom partitioning A challenge in parallelizing the subgraph discovery algorithm is that random partitioning is inefficient (as will be illustrated in Sect. 5.5). This is due to the fact that to run random walkers over partitions, a random walker needs to have access to the data in another partition to take further steps. To avoid this, our proposed partitioner first distributes rarest keyword holders across h partitions. Then, using a Breadth First Search (BFS) strategy, all the neighbors of the rarest keyword holder, and so on (up to a predefined

⁷ RDD is a fundamental data structure in Spark. RDD provides an an immutable, partitioned collection of elements in which they can be operated on in parallel.

threshold), are assigned to the same partition. Note that the threshold is a stopping point to avoid covering the entire graph for each rarest keyword holder. This increases the probability of visiting the next node of the walker in the same partition and avoids data shuffling.

Load balancing We use two strategies for load balancing. Using our custom partitioner, to avoid overlap among partitions, we assign the neighbour node to one of the desired partitions by selecting the one with the lowest cardinality (i.e., fewest nodes). In this way, each partition traces a similar search space size. Moreover, we limit the number of steps a random walker can take (as described in Sect. 4.1). Using these two strategies, the breadth of the search space is similar for different executors, and also the depth of the search space is monitored.

For each query keyword, excluding the rare keyword, a *WalkerRDD* is created and the existing subgraph members (initially the rarest keyword holder) become the restart set for the walker. Each walker retrieves the list of neighbours from its *GraphRDD* and moves to one at random. If the walker has not reached the maximum steps it can take, and the current *GraphRDD* partition contains the data for its new position, the walker keeps walking. Otherwise, the walker waits to be shuffled. Once all walkers have been terminated, the ones corresponding to a specific subgraph are located in the same partition. Then, the existing subgraphs are updated and the process is repeated for the rest of the uncovered keywords. At the end, subgraphs in *AnswerRDD* are sorted based on their scores and the best subgraphs are discovered.

4.3 Answer presentation

Aside from computational complexity, another challenge in robust keyword search is how to display the results. In addition to identifying the content nodes, we want to show how they are connected in the underlying graph, especially when the chosen content nodes are indirectly connected through middle nodes. We discuss the following two visualization options.

- *BFS* We traverse the graph in a breadth-first fashion, starting at each content node. We stop when we find the first middle node that is connected to all the content nodes. Then, all the nodes and edges that have been visited so far are presented to the user.
- *Pruned BFS* This is similar to BFS. However, we only show the nodes with high RWR scores. Note that this visualization method cannot be used with previous approaches that do not compute RWR scores.

Note that the first approach (BFS) returns more nodes to the user. For fewer input keywords, this might be the preferred choice as it reveals more connections among nodes. However, for more than three or four keywords, BFS might return many nodes, making it difficult to view the results. Therefore, for larger numbers of input keywords, Pruned BFS may be a better choice.

4.4 Relevance of answers

In this work, we use a random walk with restart (RWR) to calculate the relative importance of nodes in the entire graph. In some cases, nodes with higher degrees have higher importance scores than nodes with lower degrees. However, nodes with high degrees are not the only nodes that may have high importance according to RWR. In some situations,

a node a might obtain a significant amount of importance from its neighbour node b if b has many connections in the graph (even if a does not have many connections). This is how RWR captures the importance of nodes in the graph (Leskovec et al. 2014). For example, in co-authorship networks, an author x with many publications and co-authors has a high importance. However, a co-author of x (y) also has a high importance because they published with x , even if y does not have many co-authors. Therefore, the use of RWR provides a meaningful way of ranking nodes in the graph that takes into account recursively the entire graph structure. The same scenario happens during PageRank (which is based on a random walk process) to rank Web pages: pages that are referenced by an important page also become important (Leskovec et al. 2014).

In another scenario, researcher a publishes few important papers with few other researchers and those researchers also collaborate with few other researchers. In this case, although a is an important researcher, she might not be highly-ranked in the RWR process. This situation might also happen with the use of shortest paths: a may not appear on many shortest paths since she and her neighbors have few neighbours. One way to alleviate this issue is to assign node weights based on their importance (e.g., significance of research), and then distribute this weight onto the edges before running the RWR process. In this case, the graph becomes weighted, and the chance of the RWR surfer following an edge with a high value (which is connected to an important node) is higher. We plan to examine this approach in future work.

5 Experiments

5.1 Datasets and settings

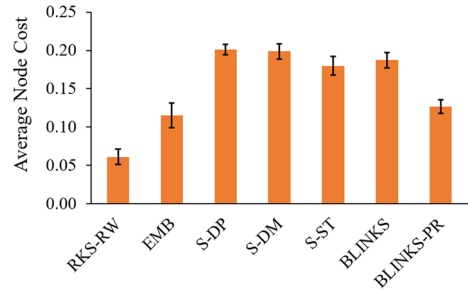
We use the DBLP⁸ and IntAct PPI⁹ graphs to evaluate our algorithms against existing approaches. The DBLP dataset contains information about authors and their publications. We form a network of authors that contains 1.3M nodes (two authors are connected if they published together in the past) and 3.95M edges. In the DBLP dataset, the maximum node degree is 473 and the average node degree is 6.1. We also compute the *h-index* of each author using ArnetMiner (Tang et al. 2008). In DBLP, if a keyword appears at least twice in the titles of an author's publications, it is considered as a keyword (or expertise) of that author. The PPI network has 70k nodes and 425k edges. In the PPI network, the maximum degree of a node is 4,019 and the average node degree is 12.1. The nodes are proteins and the edges indicates biological interactions between them. We treat genes that are associated with each protein as the keywords of the protein. We calculate the expression level of each protein based on the length and structure of the protein given in UniProt.¹⁰ In the Monte Carlo approximation, we set the number of samples to 20.

⁸ <http://dblp.uni-trier.de/xml/>.

⁹ <https://www.ebi.ac.uk/intact>.

¹⁰ <http://www.disgenet.org/>.

Fig. 5 Average node cost of answer subgraphs over the PPI network. Lower node cost corresponds to higher expression level



5.2 Comparison methods

We test the performance of our algorithm against the state of the art. All algorithms are implemented in Java.¹¹ In the remainder of this section, RKS-RW denotes our algorithm (random walk with restart). We compare it with the following methods:

- *S-DP* a shortest path-based algorithm proposed in Kargar et al. (2014). This method minimizes the sum of distances between all pairs of content nodes. The idea of this algorithm is to form a subgraph around each content node and then cover the rest of the keywords.
- *S-DM* a shortest-based algorithm proposed in Lappas et al. (2009), which introduced the problem of team formation in social networks. The objective is to minimize the diameter of the answer subgraph. In order to do this, the algorithm first selects the rarest keyword holders. Then, an answer is formed around each of these nodes and the one with the smallest diameter is selected as the best answer.
- *S-ST* a recent method published in Yin et al. (2019) that also addresses the team formation problem based on the notion of the shortest path. The objective function is inspired by the objective function of the classical Group Steiner Tree problem and a greedy algorithm is proposed to optimize this objective.
- *EMB* an embedding-based method that transforms the graph into a vector space through *node2vec* (Grover and Leskovec 2016). The distance between a pair of nodes is equal to the Euclidean distance between their respective vectors. To form answers, we use the algorithm from Lappas et al. (2009). Embeddings were recently used in Cavallari et al. (2017) to address the problem of *community search*.
- *BLINKS* a shortest-path based algorithm proposed in He et al. (2007), which minimizes the distances among nodes.
- *BLINKS-PR* the algorithm proposed in He et al. (2007), which optimizes the distances as well as the static PageRank scores of content nodes. The tradeoff parameter that incorporates the distances and the static PageRank scores is set to 0.5.

¹¹ The code is available at: <https://people.ryerson.ca/kargar/robustsearch.zip>.

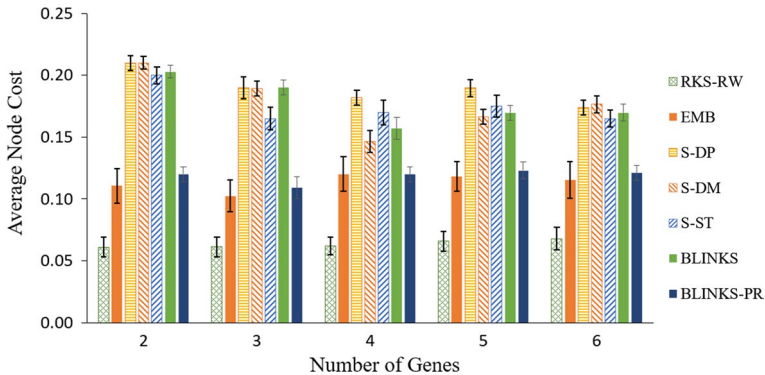
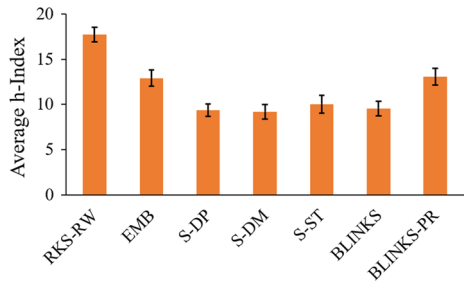


Fig. 6 Average node cost over the PPI network for different numbers of keywords (genes) in the query

Fig. 7 Average h-index of team members over the DBLP graph



5.3 Quality of answers

5.3.1 Expression level of answers over the PPI network

In this section, we calculate the protein expression levels of subgraph answers discovered by different algorithms. The expression level of each protein is determined using the method proposed in Duret and Mouchiroud (1999); Ingvarsson (2007b). A cost value is assigned to each protein; the lower the cost, the higher the expression level. A higher expression level increases the chance of interaction among proteins. The average cost of each protein in the answer determines the answer's cost. We randomly generate 100 queries with 2 to 6 genes responsible for different diseases. Figure 5 shows the cost (i.e., expression level) of answer subgraphs for different algorithms. The average cost of answers produced by our algorithm, RKS-RW, is lower than that of previous approaches. The difference is significant according to the t-test with a p value of 0.05. Figure 6 illustrates the cost of the answer subgraphs for different number of required genes (i.e., different numbers of keywords in a query). Our approach consistently performs better than previous approaches for different numbers of genes (with p -values smaller than 0.05). Note that all the shortest-path based methods (S-DP, S-DM, S-ST, and BLINKS) achieve similar results. Furthermore, the results of EMB and BLINKS-PR are close. This is because in order to transform the graph into the vector space, *node2vec* runs multiple random walks (Grover and Leskovec 2016). Therefore, the vector space models the output of a static PageRank, which is part of the scoring function of BLINKS-PR.

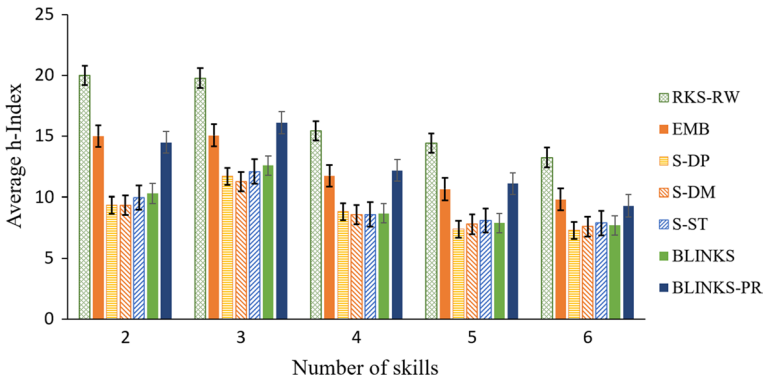
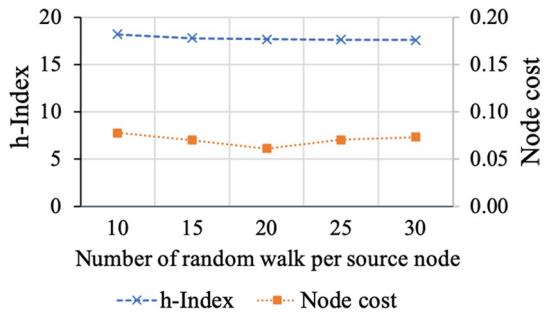


Fig. 8 Average h-index of team members over the DBLP graph vs. the number of skills (keywords) in the query

Fig. 9 Average author h-index over the DBLP graph and the average of node cost over the PPI network for different sample numbers (i.e., the number of random walks per source node) in the Monte Carlo approach



5.3.2 H-index of teams over the DBLP dataset

We now evaluate the quality of teams over the DBLP dataset in terms of the h-index of the authors. First, we randomly generate 100 skill sets (each skill set contains 2 to 6 skills). Then, we calculate the h-index of each discovered team, which is the average h-index of its members (nodes that contain at least one keyword). Figure 7 illustrates the average h-index of the teams discovered by different algorithms. RKS-RW outperforms all other methods. The difference is significant according to the t-test with a p -value of 0.05. Figure 8 shows the h-index of each algorithm for different number of required skills. Again, RKS-RW consistently generates teams with higher h-indices (with p -values smaller than 0.05).

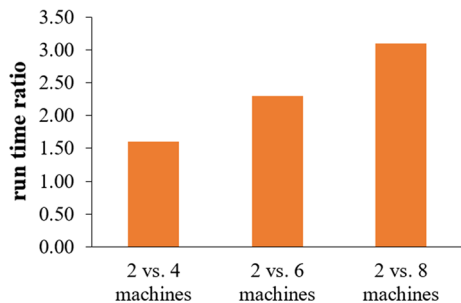
5.4 Sensitivity analysis

Next, we analyze the sensitivity of RKS-RW with respect to the number of samples in the Monte Carlo approach. We change the number of samples from 10 to 30 and observe how the node cost (for the PPI network) and the h-index (for the DBLP graph) change. Figure 9 illustrates the effect of the number of samples on the performance of RKS-RW. The average h-index of the answer subgraph is not sensitive to the number of samples. Moreover, by increasing the number of samples, we do not observe significant changes in the PPI node

Table 1 Runtime improvement of RWR-based algorithms on the DBLP dataset

	1K nodes	46K nodes	569K nodes	1.3M nodes
Monte Carlo vs. Naive	6 times faster	215 times faster	NA	NA
Distributed vs. Naive	10 times slower	43 times faster	NA	NA
Distributed vs. Monte Carlo	60 times slower	5 times slower	2.5 times faster	3.8 times faster

Faster run times are shown in bold

Fig. 10 Scalability analysis of the distributed RKS-RW algorithm

cost (i.e., protein expression level). This suggests that we do not need a large number of samples for estimating the exact random walk scores in the Monte Carlo approach.

5.5 Scalability of distributed RWR

Finally, we evaluate the scalability of the proposed distributed random walk algorithm on the DBLP dataset as it contains a large number of nodes. The experimental environment consists of one Spark master node and five Spark worker nodes. Each node has 4 cores and 16 GB of main memory. We use Spark 2.3.0 as our cluster computing framework. To test how well the proposed algorithms scale, we divide the DBLP graph into four graphs of varying sizes. To preserve the structure of the graph, the smaller graphs are created by applying a *breadth-first search* on an arbitrary node in the graph, and by increasing the depth level from 1 to 4. This process creates four graphs of increasing sizes. The smallest graph has 1K nodes, and the largest one has 1.3M nodes.

Table 1 shows the runtime improvement of the RWR algorithm on four graphs with different sizes. The algorithm we refer to as *naive* does not include any optimization for computing the RWR scores (line 10 of Algorithm 1). Note that the naive algorithm performs poorly on large graphs and does not terminate after one hour on graphs with more than 0.5M nodes. The Monte Carlo algorithm scales linearly as the number of nodes in the graph increases. Furthermore, the distributed version of the RWR algorithm significantly decrease the runtime for large graphs. For small graphs, the distributed version suffers from the distribution overhead and its runtime is longer than both naive and Monte Carlo for graphs with 1K nodes.

Distributed RKS-RW shows an almost linear scalability as the number of available machines increases over the DBLP dataset. We report the results in Fig. 10. Here, we present the ratio of the runtime of the distributed algorithm when using 2 machines versus 4 machines, and also 2 machines versus 6 machines. This ratio is over 1.5 when using 4 machines, over 2.0 when using 6 machines, and slightly over 3.0 when using 8 machines; this suggests linear scalability of our distributed algorithm.

5.6 Query discussion

We observed that for rare query keywords (i.e., keywords that appear in only few nodes in the graph), the results of our RWR-based method are closer to the results of shortest path based methods. However, by increasing keyword frequency (e.g., from 20 nodes to 100 nodes in the DBLP dataset), the results of our approach improve while the results of shortest path-based methods stay the same. The reason is that when the input keywords appear in more nodes in the graph, RWR is more likely to select candidates that have many paths to the nodes that are already selected. Furthermore, these nodes are more likely to have a higher score (e.g., higher h-index). However, for the shortest path-based methods, even with fewer keyword holders, it is still possible to form a subgraph with few nodes and edges. The embedding approach shows similar behaviour to RWR. It produces better results with keywords with higher frequency since it models the entire graph in a vector space and avoids the limitations of shortest path based methods. This observation was valid for all tested numbers of input keywords (from two to six). We also observe that queries with fewer input keywords produce slightly better results than queries with more keywords.

6 Conclusions

We formulated and studied the problem of robust keyword search in attributed graphs. We proved that the problem is NP-hard and proposed a solution based on the notion of random walk with restart (RWR). To improve the efficiency of the RWR approach, we proposed to use Monte Carlo approximation. We also developed a distributed version of our RWR approach to scale well for large graphs. Experiments on large real-life graphs verified the effectiveness of our approach compared to the previous state of the art. In future work, we plan to extend our framework to discover multi-objective optimal answers. In addition to ensuring close connections between content nodes, we will consider objectives such as node cost, or expert productivity and workload in the special case of expert networks. We also plan to extend this work to dynamic graphs.

Acknowledgements We would like to thank the anonymous reviewers for their comments that greatly improved the quality and presentation of this manuscript.

References


- Anagnostopoulos, A., Becchetti, L., Castillo, C., Gionis, A., & Leonardi, S. (2012). Online team formation in social networks. In *WWW '12*, pp. 839–848.
- Bahmani, B., Chowdhury, A., & Goel, A. (2010). Fast incremental and personalized pagerank. *Proceedings of the VLDB Endowment*, 4(3), 173–184.

- Bai, L., Liang, J., Du, H., & Guo, Y. (2018). A novel community detection algorithm based on simplification of complex networks. *Knowledge-Based Systems*, 143, 58–64.
- Balmin, A., Hristidis, V., & Papakonstantinou, Y. (2004). Objectrank: Authority-based keyword search in databases. In *VLDB*, pp. 564–575.
- Bhalotia, Gaurav, Hulgeri, Arvind, Nakhe, Arvind, Chakrabarti, Soumen, & Sudarshan, Shashank. (2002). Keyword searching and browsing in databases using banks. In *Proceedings 18th international conference on data engineering*, pp. 431–440. IEEE.
- Cavallari, S., Zheng, V., Cai, H., Chang, K., & Cambria, E. (2017). Learning community embedding with community detection and node embedding on graphs. In *CIKM*, pp. 337–386.
- Chakrabarti, S. (2007). Dynamic personalized pagerank in entity relation graphs. In *WWW*, pp. 571–580.
- Cui, W., Xiao, Y., Wang, H., & Wang, W. (2014). Local search of communities in large graphs. In *SIGMOD*, pp. 991–1002.
- Ding, B., Yu, J., Wang, S., Qin, L., Zhang, X., & Lin, X. (2007). Finding top-k min-cost connected trees in databases. In *ICDE*, pp. 836–845.
- Ding, B., Yu, J. X., Wang, S., Qin, L., Zhang, X., & Lin, X. (2007). Finding top-k min-cost connected trees in databases. In *2007 IEEE 23rd international conference on data engineering*, pp. 836–845. IEEE.
- Duret, L., & Mouchiroud, D. (1999). Expression pattern and surprisingly, gene length shape codon usage in caenorhabditis, drosophila, and arabidopsis. *Proceedings of the National Academy of Sciences*, 13(96), 4482–4487.
- Duret, L., & Mouchiroud, D. (1999). Expression pattern and, surprisingly, gene length shape codon usage in caenorhabditis, drosophila, and arabidopsis. In *Proceedings of the national academy of sciences of the United States of America*, pp. 4482–4487.
- Fang, Y., Cheng, R., Luo, S., & Hu, J. (2016). Effective community search for large attributed graphs. *PVLDB*, 9(12), 1233–1244.
- Gonzalez, M., & Kann, M. (2012). Protein interactions and disease. *PLoS Computational Biology*, 8(12), 819–830.
- Grover, A., & Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 855–864. ACM.
- Hajiabadi, M., Zare, H., & Bobarshad, H. (2017). IEDC: An integrated approach for overlapping and non-overlapping community detection. *Knowledge-Based Systems*, 123, 188–199.
- Han, S., Zou, L., Yu, J.X., & Zhao, D. (2017). Keyword search on RDF graphs: A query graph assembly approach. In *CIKM*, pp. 227–236.
- He, H., Wang, H., Yang, J., & Yu, P. (2007). BLINKS: Ranked keyword searches on graphs. In *SIGMOD*, pp. 305–316.
- Huang, X., & Lakshmanan, L. V. S. (2017). Attribute-driven community search. *PVLDB*, 10(9), 949–960.
- Ingvarsson, P. (2007). Gene expression and protein length influence codon usage and rates of sequence evolution in populus tremula. *Molecular Biology and Evolution*, 24(3), 836–844.
- Ingvarsson, Pär K. (2007). Gene expression and protein length influence codon usage and rates of sequence evolution in populus tremula. *Molecular Biology and Evolution*, 24(3), 836–844.
- Jay, S. M. (2019). Protein silencing to stop a silent killer. *Science Translational Medicine*, 11(473), 529.
- Jeong, Y., Pan, Y., Rathore, S., Kim, B., & Park, J. H. (2019). A parallel team formation approach using crowd intelligence from social network. *Computers in Human Behavior*, 101, 429–434.
- Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., & Karambelkar, H. (2005). Bidirectional expansion for keyword search on graph databases. In *VLDB*, pp. 505–516.
- Kargar, M., & An, A. (2011). Keyword search in graphs: Finding r-cliques. *PVLDB*, 10(4), 681–692.
- Kargar, M., An, A., & Yu, X. (2014). Efficient duplication free and minimal keyword search in graphs. *TKDE*, 26(7), 1657–1669.
- Kargar, M., Golab, L., & Szlichta, J. (2016). eGraphSearch: Effective keyword search in graphs. In *CIKM, system demonstration*, pp. 2461–2464.
- Kargar, M., Zihayat, M., & An, A. (2013). Finding affordable and collaborative teams from a network of experts. In *SDM*, pp. 587–595.
- Kasneci, G., Ramanath, M., & Sozio, M. (2009). STAR: Steiner-tree approximation in relationship graphs. In *ICDE*, pp. 868–879.
- Lao, N., & Cohen, W. (2010). Fast query execution for retrieval models based on path-constrained random walks. In *KDD*, pp. 881–888.
- Lappas, T., Liu, L., & Terzi, E. (2009). Finding a team of experts in social networks. In *KDD*, pp. 467–476.
- Le, W., Li, F., Kementsietsidis, A., & Duan, S. (2014). Scalable keyword search on large RDF data. *TKDE*, 26(11), 2774–2788.

- Leskovec, J., Rajaraman, A., & Ullman, J. (2014). *Mining of massive datasets* (2nd ed.). Cambridge: Cambridge University Press.
- Li, G., Ooi, B., Feng, J., Wang, J., & Zhou, L. (2008). EASE: Efficient and adaptive keyword search on unstructured, semi-structured and structured data. In *SIGMOD*, pp. 903–904.
- Li, R. H., Qin, L., Yu, J. X., & Mao, R. (2015). Influential community search in large networks. *PVLDB*, 8(5), 509–520.
- Majumder, A., Datta, S., & Naidu, K. (2012). Capacitated team formation problem on social networks. In *KDD*, pp. 1005–1013.
- Markowetz, A., Ying, Y., & Papadias, D. (2007). Keyword search on relational data streams. In *SIGMOD*, pp. 605–616.
- Pinero, J. (2017). DisGeNET: A comprehensive platform integrating information on human disease-associated genes and variants. *Nucleic Acids Research*, 45(D1), 833–839.
- Prud'hommeaux, E. (2008). SPARQL query language for RDF, W3C recommendation. <http://www.w3.org/TR/rdf-sparql-query/>.
- Qin, L., Yu, J., Chang, L., & Tao, Y. (2009). Querying communities in relational databases. In *ICDE*, pp. 724–735.
- Sozio, M., & Gionis, A. (2010). The community-search problem and how to plan a successful cocktail party. In *KDD*, pp. 939–948.
- Tang, J., Zhang, J., Yao, L., Li, J., Zhang, L., & Su, Z. (2008). Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 990–998. ACM.
- Termehchy, A., & Winslett, M. (2011). Using structural information in XML keyword search effectively. *ACM Transactions on Database Systems (TODS)*, 36(2), 1–49.
- Wang, H., & Aggarwal, C. C. (2010). *A survey of algorithms for keyword search on graph data*, (pp. 249–273). Springer US, Boston, MA.
- Wang, W., He, Z., Shi, P., Wu, W., Jiang, Y., An, B., et al. (2019). Strategic social team crowdsourcing: Forming a team of truthful workers for crowdsourcing in social networks. *IEEE Transactions on Mobile Computing*, 18(6), 1419–1432.
- Yang, Y., Agrawal, D., Jagadish, H. V., Tung, A. K. H., & Wu, S. (2019). An efficient parallel keyword search engine on knowledge graphs. In *2019 IEEE 35th international conference on data engineering (ICDE)*, pp. 338–349.
- Yin, X., Qu, C., Wang, Q., Wu, F., Liu, B., Chen, F., et al. (2019). Social connection aware team formation for participatory tasks. *IEEE Access*, 6, 20309–20319.
- Zhang, J., Yu, P. S., & Lv, Y. (2017). Enterprise employee training via project team formation. In *WSDM*, pp. 3–12.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Affiliations

Spencer Bryson¹ · Heidar Davoudi¹ · Lukasz Golab² · Mehdi Kargar³  · Yuliya Lytvyn² · Piotr Mierzejewski⁴ · Jaroslaw Szlichta^{1,4} · Morteza Zihayat^{3,4}

Spencer Bryson
bryson@ontariotechu.ca

Heidar Davoudi
davoudi@ontariotechu.ca

Lukasz Golab
lgolab@uwaterloo.ca

Yuliya Lytvyn
ylytvyn@uwaterloo.ca

Piotr Mierzejewski
piotrm@ca.ibm.com

Jaroslav Szlichta
jarek@ontariotechu.ca

- ¹ Ontario Tech University, Oshawa, Canada
- ² University of Waterloo, Waterloo, Canada
- ³ Ted Rogers School of Management, Ryerson University, Toronto, Canada
- ⁴ IBM Centre for Advanced Studies, Markham, Canada