Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Efficient keyword search over graph-structured data based on minimal covered *r*-cliques

Asieh GHANBARPOUR[1,2], Khashayar NIKNAFS[1], Hassan NADERI[†‡1]

[1]*School of Computer Engineering, Iran University of Science and Technology, Tehran 13114-16846, Iran*

[2]*Department of Computer Engineering, University of Sistan and Baluchestan, Zahedan 98167-45845, Iran*

[†]E-mail: naderi@iust.ac.ir

Received Mar. 3, 2018; Revision accepted Sept. 11, 2018; Crosschecked Mar. 5, 2020

Abstract: Keyword search is an alternative for structured languages in querying graph-structured data. A result to a keyword query is a connected structure covering all or part of the queried keywords. The textual coverage and structural compactness have been known as the two main properties of a relevant result to a keyword query. Many previous works examined these properties after retrieving all of the candidate results using a ranking function in a comparative manner. However, this needs a time-consuming search process, which is not appropriate for an interactive system in which the user expects results in the least possible time. This problem has been addressed in recent works by confining the shape of results to examine their coverage and compactness during the search. However, these methods still suffer from the existence of redundant nodes in the retrieved results. In this paper, we introduce the semantic of minimal covered *r*-clique (MCC$_r$) for the results of a keyword query as an extended model of existing definitions. We propose some efficient algorithms to detect the MCC$_r$s of a given query. These algorithms can retrieve a comprehensive set of non-duplicate MCC$_r$s in response to a keyword query. In addition, these algorithms can be executed in a distributive manner, which makes them outstanding in the field of keyword search. We also propose the approximate versions of these algorithms to retrieve the top-k approximate MCC$_r$s in a polynomial delay. It is proved that the approximate algorithms can retrieve results in two-approximation. Extensive experiments on two real-world datasets confirm the efficiency and effectiveness of the proposed algorithms.

Key words: Keyword search; Graph mining; Information retrieval; Database; Clique
https://doi.org/10.1631/FITEE.1800133 **CLC number:** TP391

1 Introduction

Keyword search is an alternative for structured languages in querying over graph-shaped structured and semi-structured data such as relational databases, XML databases, and RDF databases. Keyword search is proposed with respect to the convenience of common users in querying datasets. In this type of search, the user expresses his/her query by entering just some keywords without knowing the underlying schema of data or having familiarity with the complex syntax of a structured query language.

Since the queries in keyword search are syntax-free, the semantic of the query should be extracted from the existing graph data. A result to a keyword query is a compact connected structure which covers all or part of the queried keywords. As an example, Fig. 1a shows a part of the DBLP dataset. Suppose that keyword query Q={Ba, Liu, 2016} has been expressed on this graph. Fig. 1b shows three results to this query. Each of results A_1 and A_2 shows a common paper written by "Ba" and "Liu," which was published in 2016. Similarly, A_3 shows the paper "Bluetooth low energy," which was written by Ba and Liu, and refers to paper "Fully integrated Bluetooth," which was also written by the two authors. Although

‡ Corresponding author
ORCID: Hassan NADERI, https://orcid.org/0000-0002-3296-8505

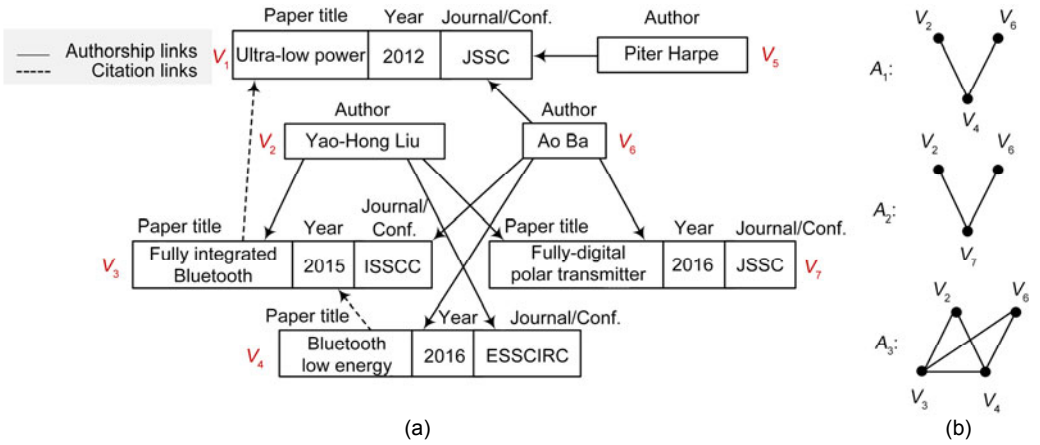

Fig. 1 A part of the DBLP dataset (a) and some of results to query Q={Ba, Liu, 2016} (b)

this result is also relevant to the query, it provides some additional information which may not be significant for the user.

In the literature, the following three conditions have been considered as the basis for ranking the results of a keyword query:

1. coverage of queried keywords (the result which covers a greater number of queried keywords is ranked higher),

2. closeness of queried keywords, and

3. minimality of results (a result is minimal if it does not contain any subtree/subgraph that also satisfies the result conditions).

In these works, the results are retrieved based on different strategies and then ranked based on the mentioned properties. This implies that all the relevant results should be retrieved before ranking. To have more efficient algorithms, the desired properties of results should be checked sooner and in the search step. To this goal, the concepts of Steiner tree, distinct root-based semantics, Steiner graph, and *r*-clique are proposed as the desired results that should be searched. A Steiner tree considered in Kimelfeld and Sagiv (2006), Ding et al. (2007), and Hao et al. (2015) covers all the given keywords with the minimum sum of edge weights. Since the problem of finding Steiner trees in a graph is an NP-hard problem (He et al., 2007), distinct root-based semantics was proposed to define the results of keyword queries. According to this semantics, a result covers all given keywords with minimum central distances to a root node. However, there are two problems with this. First, it is aimed mainly at compacting the structure of results based on the closeness of their nodes to a central node, while this does not guarantee the closeness of keyword nodes (the nodes covering at least one of the queried keywords). Second, using the distinct root-based semantics, the root of retrieved results would be distinct. This means losing many of the results with the same root of the retrieved results. The recent works in keyword search tend to find subgraphs instead of subtrees because they are more informative. The concepts of *r*-radius Steiner graph (Li et al., 2008) and *r*-clique (Kargar et al., 2014; Kargar and An, 2015) are for retrieving the compact subgraphs as the results of keyword queries. The coverage of keywords and the closeness of keyword nodes are well embedded in these concepts. However, these definitions do not emphasize the minimality of results.

In this study, we propose to find minimal covered *r*-cliques (MCC$_r$s) of a graph as the results of a keyword query. An MCC$_r$ is a compact structure of vertices with three properties: (1) It covers all the queried keywords; (2) The distance between every pair of its vertices is no larger than r; (3) It does not have a proper subgraph that is also an MCC$_r$. An MCC$_r$ is a compact and meaningful structure which contains some relevant intermediate nodes for improving the answerability. Moreover, MCC$_r$ is more concise than the previous semantics since it contains no irrelevant nodes.

The precise form of MCC$_r$s makes identifying them in a large graph difficult. In this paper, we introduce some algorithms to identify the MCC$_r$s of a keyword query in efficient ways. The proposed algorithms employ effective pruning techniques to reduce

the time complexity of search. The main contributions of this paper are summarized as follows:

1. We propose a new semantics for the results of a keyword search in which the keyword vertices are close to each other.

2. We present the Bron-Kerbosch-based keyword search algorithm (BKS) as the base algorithm for retrieving all the MCCrs of a homogenous graph, in which pruning methods are employed as in the Bron-Kerbosch algorithm (Bron and Kerbosch, 1973). BKS uses pivoting and type restrictions to reduce the exploration cost of finding MCC_rs.

3. Algorithm BKSR is proposed as an improved version of BKS. This algorithm searches the graph more efficiently by imposing a frequency-based ordering on the graph's vertices.

4. We propose BKSM and BKSRM as distributed versions of BKS and BKSR algorithms, respectively. These algorithms rely on the distributive nature of the base algorithms for parallel search of the search space.

5. A heuristic algorithm is presented to produce top-k results of a keyword query in a polynomial delay. This algorithm can incrementally generate results by observing the distance restrictions. Using this algorithm, an approximate version is proposed for each of the exact algorithms.

6. A set of extensive experiments is conducted on two real datasets IMDb and DBLP with a comparison to the existing algorithms to show the efficiency and effectiveness of the proposed algorithms.

2 Related works

Research on keyword search can be followed in four categories: keyword search on relational databases (Liu et al., 2006; Ding et al., 2007; Park J and Lee, 2011; Bergamaschi et al., 2013, 2016; Xu et al., 2013), keyword search on XML databases (Guo et al., 2003; Le et al., 2015), keyword search on the Semantic Web (Ning et al., 2009; Wang et al., 2015), and keyword search on schema-free graphs (Ghanbarpour and Naderi, 2018). All the relational, XML, and RDF datasets are associated with the predefined schemas. The existence of a schema does not impose any restriction on the graph, and facilitates the determination of the query meaning. However, there are many graphs for which the schema is not defined. Keyword search over schema-free graphs has been widely examined in the literature (He et al., 2007; Golenberg et al., 2008; Kim et al., 2012; Yuan et al., 2013; Park CS and Lim, 2015). Even though these works provide a general framework for keyword search over any type of graph, they face more challenges because of having less prior knowledge about the examined data. In the following, some of these works are mentioned.

BLINKS (He et al., 2007) partitions graph data into some blocks and uses a bi-level indexing to process keyword queries. One group of indices is used to travel between the blocks and the other is used to access the data within the blocks. In retrieving an answer, BLINKs begins from a keyword node and searches its block with the help of intra-block indices. If searching for an answer is expanded to the neighborhood blocks, it uses multiple cursors and sends each of them to a neighbor block to continue the search. According to the claim of the authors, BLINKS is m-optimal, where m is the number of input keywords. Ease, which was introduced in Li et al. (2008), defines the concept of r-radius subgraphs on an undirected graph and provides a method to group them. It uses two indices for storing data: the first index saves the structural distance between each pair of keywords and the second index stores the contained r-radius subgraphs for each keyword. In this approach, the size of the final answers is limited to the size of the initial processed subgraphs. So, there may be answers that have not been found. This approach contains a ranking step to sort retrieved answers according to their structural compression and some information retrieval measures. LABRADOR (Mesquita et al., 2007), as an extension of the system proposed by Calado et al. (2004), is a keyword search system over attributed graphs. It uses Bayesian-based probabilities to estimate the relevance of answers to a given query. After retrieval, the candidate answers are ranked according to the probability that they have of representing the best assignment between the keywords provided by the user and the attributes of the database schema. de Virgilio et al. (2009) proposed an approach based on the paths which led to the keyword nodes. In this approach, a path of graph is selected in each step to be added to an incomplete answer. The paths are grouped based on their template and assigned with a score. The score of paths is used not only in the expanding phase, but also in the final

ranking of answers.

Kargar and An (2011) focused on finding answers in the form of *r*-cliques. An *r*-clique is a set of nodes which covers all the query keywords and whose shortest path between every two nodes is not greater than *r*. *r* is a user-defined parameter and indicates the maximum value of the answer's diameter. Two approximate algorithms *r*-clique and *r*-clique-rare were proposed in this work to find results of keyword queries. In both algorithms, the search space is divided based on the Lawler algorithm and the answers are detected in the distinct subspaces. In the *r*-clique algorithm, all the covered star-shaped structures with different keywords as the roots are examined, and the one with the maximum weight is selected as the best result in each subspace. The authors claimed that, in the worst case, the weight of an answer produced by their algorithm is twice the weight of the optimal answer. However, since the time complexity of this algorithm is relatively high, they proposed algorithm *r*-clique-rare. In this algorithm, instead of considering all the star-shaped structures, only the ones with the rarest nodes in their roots are considered. Although *r*-clique-rare is faster than *r*-clique, the accuracy of its results is lower. Kargar et al. (2014) proposed a Lawler-based method to retrieve the top-*k* non-duplicate answers. The duplicate answers are the ones covering the same set of keyword nodes, although these nodes may be connected differently.

Nguyen and Cao (2012) presented an approach by selecting the top-*k* data sources from potentially numerous data sources. Their method derives information patterns from each data source as succinct synopses that act as representatives of the corresponding data sources. The patterns (subgraphs) are then scored based on their relevance to the given query using a structure-aware ranking function. A new type of keyword search query, ROU-query, was defined in Pan and Wu (2013). It uses input keywords in three categories, required, optional, and unwanted, and returns nodes of the underlying graph whose neighborhood satisfies the keyword requirements. It applies a new data structure named the query induced partite graph (QuIP) to capture the constraints related to the neighborhood size and unwanted keywords. The authors proposed a family of algorithms which take advantage of the information in QuIP for efficient evaluation of ROU-queries.

The problem of finding duplication-free and minimal answers was addressed in Kargar et al. (2014), and an algorithm on the basis of Lawler's procedure was proposed as a solution. Park CS and Lim (2015) proposed an approach to aggregate the best keyword nodes gained from a pre-computed process in order to produce the top-*k* relevant answers in an approximate order. They used a queuing system over the data extracted from an extended inverted index. Their method prefers more extended and relevant answers having more coverage of keywords instead of minimal answers.

3 Preliminaries and problem statement

Given a keyword query with l keywords as $Q=\{k_1, k_2, \ldots, k_l\}$ over a data graph, the problem of keyword search in the graph is to find a set of connected structures as results, where each of them satisfies the keyword query. The results of a keyword query can be in the form of subtrees or subgraphs. According to the "AND semantic," any result should cover all of the queried keywords and, based on the "OR semantic," any result should cover at least one queried keyword. In this work, relying on the AND semantic, we define the MCC_r as a result to a keyword query. An MCC_r is structurally compact and textually covers all the queried keywords.

Definition 1 (Covered *r*-clique) A covered *r*-clique C of graph G is a set of connected keyword vertices in G that together cover all of the queried keywords and in which the shortest distance between any pair of the keyword vertices is equal to or lower than r (Kargar and An, 2015).

Definition 2 (Minimal covered *r*-clique) A covered clique C of G is minimal if there exists no covered clique C' in G such that C' is a subset of C.

Retrieving MCC_r as the result of a keyword query has two advantages: (1) The semantics of MCC_r guarantees that the inside distances between any two keyword nodes would not be larger than r. It shows the structural compactness of the retrieved results. (2) An MCC_r is a specific subgraph that can be detected in a graph more efficiently than subgraphs with no predefined shapes.

Since there may be numerous results to a keyword query, it is expected that results will be

presented in a ranked order based on their weights. We use the sum-based semantic to define the weight of an MCC_r.

Definition 3 (Weight of MCC_r) According to the sum-based semantic, the weight of an MCC_r of G retrieved in response to query Q with l keywords is calculated as follows:

$$\text{Weight}=\sum_{i=1}^{l}\sum_{j=i+1}^{l}\text{dist}(v_i,v_j), \tag{1}$$

where v_i and v_j are vertices of MCC_r, and dist(v_i, v_j) shows the shortest distance between v_i and v_j in G. In this work, the results with smaller weights are ranked higher.

In Kargar et al. (2014) and Kargar and An (2015), it was shown that finding covered r-cliques with the minimum weight (based on Eq. (1)) is an NP-hard problem. By a reduction of MCC_r to covered r-clique, it is easily proved that finding the minimum-weight MCC_r is also an NP-hard problem.

4 The proposed algorithms

In this section, we present some algorithms to find MCC_r to a keyword query. In the first algorithm, the MCC_rs are detected based on a classification on the graph's vertices based on the covered keywords. The idea of this algorithm is derived from the Bron-Kerbosch (Bron and Kerbosch, 1973) and Tomita (Tomita et al., 2006) algorithms, which have been proposed to find the maximal cliques of a homogeneous graph. However, in this work, we aim to find the MCC_r of a heterogeneous graph. The proposed algorithm uses a recursive procedure to investigate the search space by building search trees rooted at different vertices. In this procedure, the branches of search trees are pruned based on the keywords of the given query. In the improved version of this algorithm, for better performance, the range of pruning the search space is extended by maintaining an order on the keyword vertices. The next two algorithms concentrate on distributed finding of MCC_rs in the graph. Then detecting results in an approximate order is handled.

To solve the keyword search problem by finding minimal covered cliques, the graph G is augmented with an edge between any two connected nodes of G. This augmentation can be achieved in an efficient way by considering the following two optimizations:

1. Since the keywords are the base of search, just the connections to the keyword vertices are added to the graph. To this goal, a breadth-first search is performed over any keyword vertex to recognize its connections.

2. In keyword search, the results in which the vertices are far away from each other are not worth much. Therefore, the graph is augmented only by connecting the pairs of vertices whose distance is lower than a specified threshold.

The problem of detecting minimal covered cliques in the augmented graph is mapped to the problem of finding covered cliques.

Theorem 1 The set of MCC_r related to query $Q=\{k_1, k_2, \ldots, k_l\}$ in graph G is equivalent to the set of covered cliques in subgraph $g\subseteq G(V, E)$ such that $V(g)=\{\forall u\in V(G)|u\cap Q\neq\varnothing\}$ and $E(g)$ is a set of edges connecting any $v_i, v_j\in V(g)$ whose distance is no larger than r, when ignoring the possible edges between the covering vertices of the same keyword.

Proof Since there are at most l groups of vertices in g covering distinct keywords and any possible edges between vertices of a group are ignored, the number of vertices of covered cliques is at most l. With regard to the MCC_r definition and according to the facts that the number of distinct vertices in graph G is at most l and that there are no connections in G between the vertices with a distance greater than r, the set of covered cliques of l keywords in G is equivalent to the set of MCC_r of l distinct keywords in $G(V, E)$.

Using Theorem 1, the problem of solving a keyword query in graph G is reduced to the problem of finding the covered r-cliques in subgraph $g(V, E)$ containing those nodes of G with at least one given keyword being covered. All the algorithms proposed in this study solve the latter problem.

4.1 Finding all non-duplicate MCC_rs (BKS algorithm)

Finding cliques has been widely studied in the domain of graph mining (Bron and Kerbosch, 1973; Tomita et al., 2006; Segundo et al., 2016, 2018; Yu and Liu, 2017). However, this group of methods generates cliques with any size and does not impose

any restriction on the type of vertices. In this subsection, we propose BKS with the idea derived from the Bron-Kerbosch and Tomita algorithms to detect all the non-duplicate MCC$_r$s of a given query with keywords. In BKS, a number of combination trees of keywords are searched based on the depth-first search strategy. During the search, a large set of unessential branches of combination trees is pruned for efficiency. Suppose that the set of vertices covering keyword $k_i \in Q$ is shown by C_i. In BKS, two sets of vertices are considered in each step: the set of candidate vertices, P, which can be added to the MCC$_r$ that are being built, and the set R of vertices which are already added to the MCC$_r$. The algorithm begins by letting R be an empty set, and expands R level by level using a recursive procedure to reach a complete MCC$_r$. In each recursion, a vertex v of candidate vertices is selected to be added to R, and the set P is restricted to the neighbors of v. In addition, we restrict the set of candidate vertices of P in each level of expansion to $C_i \cap P$. Therefore, if the set R contains $\{v_1, v_2, \ldots, v_{i-1}\}$ in the entrance of level i, the set of candidate vertices would be

$$P = \Gamma(v_1) \cap \Gamma(v_2) \cap \ldots \cap \Gamma(v_{i-1}) \cap C_i, \qquad (2)$$

where $\Gamma(v)$ shows the neighbor set of vertex v. This setting retains the connectivity of vertices in the MCC$_r$. The recursion would not be followed if the number of vertices added to R plus that of candidate vertices is not greater than l. This is owing to the essential need to present the vertices of all types of keywords in the MCC$_r$. The pseudo code of BKS is shown in Algorithm 1. The pivoting step (according to the Tomita algorithm) in statement five of procedure BKS is to develop the range of pruning when searching the graph.

Algorithm 1 BKS algorithm

Input: input graph G, query $\{k_1, k_2, \ldots, k_l\}$
Output: MCC$_r$s

```
1  for i←1 to l do
2      C_i←set of nodes in G containing k_i
3      P←P∪C_i
4  end for
5  i←1
6  R←{}
7  BKS(P, R, i)
8  return
```

Proc BKS(P, R, i)

```
1  if P={} then
2      report R as an answer
3  end if
4  i←i+1
5  choose a pivot u∈C_i∩P
   // choose u to maximize |(C_i∩P)∩Γ(u)|
6  for each vertex v∈(C_i∩P)\Γ(u) do
7      if size ((P∩Γ(v))∪(R∪{v}))≥l then
8          BKS(P∩Γ(v), R∪{v}, i)
9      end if
10     P←P\{v}
11 end for
12 return
```

Theorem 2 Algorithm BKS generates all and only the MCC$_r$ of l keywords with no duplication.

Proof See the Appendix.

Theorem 3 The time complexity of algorithm BKS is $O(|C_{\max}|^{l+1})$, where l shows the size of the given query and $|C_{\max}|$ the maximum size of vertex set C_i containing keyword k_i for $1 \le i \le l$.

Proof The height of recursive trees of the BKS algorithm is at most l, each level associated to the vertex set of a keyword, i.e., $C_i \cap P$. Since $|C_i| \le |C_{\max}|$ for any $1 \le i \le l$, the number of times of calling procedure BKS is at most equal to $|C_{\max}|^l$. For any of recursive calls of BKS, the time required for calculating $(C_i \cap P) \cap \Gamma(u)$ is equal to the maximum size of $(C_i \cap P)$, which is $|C_{\max}|$ multiplied by $O(1)$, needed for calculating the intersection of $\Gamma(u)$ and any vertices of $(C_i \cap P)$ using the bit intersection proposed in Dasari et al. (2014). Therefore, the time complexity of BKS is obtained as $O(|C_{\max}|^{l+1})$ according to Eq. (3):

$$\left|C_{\max}\right|^l \left|C_{\max}\right| = \left|C_{\max}\right|^{l+1}. \qquad (3)$$

Note that BKS is used just as a baseline to compare with the polynomial delay approximation algorithm, which will be discussed in the next subsections. After ranking the results generated by BKS, an exact and optimal list of results would exist for any given query.

The most similar works to ours are Kargar and An (2011, 2015), which aim to find all the r-cliques (a similar concept to MCC$_r$) of a graph as the results of a given query, using a branch and bound algorithm with time complexity $O(l^2|C_{\max}|^{l+1})$. Fig. 2 shows an

example graph and the schema of traversing this graph with the Kargar-An and BKS algorithms. The BKS algorithm has several advantages over the Kargar-An algorithm: (1) The time complexity of BKS is lower; (2) BKS is an incremental algorithm which can incrementally present results before terminating the algorithm, while the Kargar-An algorithm can present results just after being terminated; (3) BKS can be easily adapted to run on a distributed framework, while the Kargar-An algorithm does not have this ability.

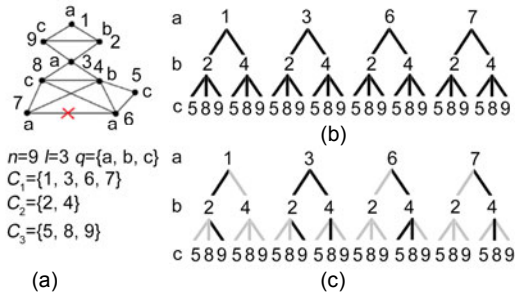

Fig. 2 An example data graph (a), schema of traversing the graph by *r*-clique (b), and schema of traversing the graph by BKS (c)

4.2 Using the frequency of keywords in MCC$_r$ detection (BKSR algorithm)

In the recursive trees of BKS, it is observed that the pruning is more in the lower levels. Therefore, the efficiency of BKS can be improved by delaying the examination of high-frequency keywords to the lower levels of recursion trees. In BKSR, the vertex set of keywords is arranged in increasing order of size and injected to the recursive calls of the algorithm based on this order. In this way, the roots of recursive trees are keyword vertices covering the less frequent keywords of the query. Using this strategy, the number of recursive trees decreases (or remains unchanged), and the number of candidates in the lower levels of the tree increases (or remains unchanged), resulting in a wider range of branches for the pruning. Fig. 3 shows the schema of traversing the graph represented in Fig. 2a by algorithm BKSR.

4.3 Distributed finding of MCC$_r$s

To increase the efficiency of the algorithms, especially in the large volume graphs, the capabilities of parallel processing can be employed. In BKS and BKSR algorithms, the recursive trees are built completely independent of each other. Therefore, they can be built in parallel. This is the base idea of BKSM and BKSRM algorithms, to parallel detection of MCC$_r$s under a distributed shared memory architecture. In these algorithms, to increase the degree of parallelism, the larger set of C_i is selected as the entries of the first level of recursion, resulting in more independent recursive trees. However, in other levels of recursion, the vertex sets are injected the same as in the base algorithms. The isolated recursive trees are executed in parallel. The pseudo code of BKSRM is shown in Algorithm 2. In this algorithm, the injection of vertex sets to each level of recursion (except the first level) is based on ascending order of their size to maintain a high possibility of pruning.

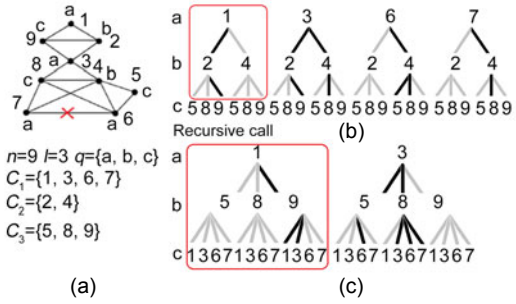

Fig. 3 An example data graph (a), schema of traversing the graph by BKS (b), and schema of traversing the graph by BKSR (c)

Algorithm 2 BKSRM algorithm

Input: input graph G, query $\{k_1, k_2, \ldots, k_l\}$
Output: MCC$_r$s

```
1  for i←1 to l do
2      C_i←set of nodes in G containing k_i
3      order_i←C_i
4  end for
5  sort order_i in ascending order based on the size of C_i
6  for i←1 to l−1 do
7      P←P∪order_i
8  end for
9  for j←1 to size(order_l) do
10     i←1
11     R←{order_l^j}
12     P←P∩Γ(order_l^j)
13     process(j, BKS(P, R, i))
14 end for
15 return
```

Theorem 4 BKSRM detects all of the covered cliques of l queried keywords with T parallel

processors for $1\leq T\leq|C_{\max}|$ with time complexity $O(|C_{\max}|^{l+1}/T)$, where $|C_{\max}|$ shows the maximum size of C_i containing keyword k_i for $1\leq i\leq l$.

Proof According to algorithm BKSRM, the recursive procedure is called at most for $(l-1)$ levels, and each level is associated to the vertex set of one of the queried keywords except the most frequent one. Therefore, the number of recursive calls would be equal to $|C_{\max}|^{l-1}$. For any recursive call, $(C_i\cap P)\cap\Gamma(u)$ should be calculated. This needs at most $O(|C_{\max}|)$. Therefore, processing each recursive tree would take at most $O(|C_{\max}|^{l})$. In the vertex set related to the more frequent keyword, there are at most $|C_{\max}|$ vertices, each of them considered as the root of a recursive tree. These vertices are assigned to different processors to be investigated in parallel. By considering the T parallel processors, the time complexity of BKSRM would be at most

$$\frac{|C_{\max}|}{T}O(|C_{\max}|^{l})=\frac{O(|C_{\max}|^{l+1})}{T}. \tag{4}$$

Since the degree of locality in the processing of the graph is high in these algorithms, they can be efficiently adapted to run on a large-scale graph processing system such as ExPregel (Sagharichian et al., 2015).

4.4 Top-*k* polynomial delay algorithm

One of the main properties of a keyword search engine is its efficiency in incrementally producing the top-k results to a given keyword query. The complexity of delay between producing two subsequent answers is a standard measure to estimate the efficiency of the search. If this delay is polynomial based on the size of the given query, the search algorithm is called a polynomial delay algorithm (Golenberg et al., 2008). In this subsection, a polynomial delay algorithm is proposed to incrementally enumerate the top-k results (MCC_r) of a given query. The general framework of this algorithm is based on the Lawler-based strategy to divide the search space. The results of each subspace are retrieved by an adaptation of the BKS algorithm. The low-weight results among those retrieved are presented as the top-k best results.

The basic Lawler algorithm (Lawler, 1972) was introduced to compute the top-k solutions to discrete optimization problems. This algorithm was then adapted to the keyword search problem (Golenberg et al., 2008; Kargar and An, 2011, 2015; Mass and Sagiv, 2012; Kargar et al., 2014). According to the adopted Lawler-based strategy (Kargar and An, 2011), the search space is divided into some overlapping subspaces proportional to the number of query keywords. Assume that the given query contains three keywords $Q=\{k_1, k_2, k_3\}$ and that the set of vertices covering any k_i is denoted by C_i. Therefore, $C_1\times C_2\times C_3$ shows the initial search space. The best result of this space is detected and added to a priority queue. Suppose that the best result is shown by $\{v_1, v_2, v_3\}$ such that v_i contains keyword k_i. According to this result, the initial search space is divided into four subspaces (Table 1). After dividing the search space, the algorithm for finding the best result is executed in each of the subspaces except SB_0. The detection results are added to the priority queue. At the end of this step, the lowest weight element of the priority queue is presented as the next best global result. The subspace to which the last result belongs is further divided to the sub-subspaces in a similar way, as shown in Table 1.

Table 1 Dividing the search space*

Subspace	Representative set
SB_0	$\{v_1\}\times\{v_2\}\times\{v_3\}$
SB_1	$(C_1-\{v_1\})\times C_2\times C_3$
SB_2	$\{v_1\}\times(C_2-\{v_2\})\times C_3$
SB_3	$\{v_1\}\times\{v_2\}\times(C_3-\{v_3\})$

* Kargar and An (2015)

The best results of sub-subspaces are also added to the priority queue and the lowest weight of the queue's elements is presented as the next best global result. This process continues until presenting k results. We use Algorithm 3, which is a modified version of Algorithm 2 presented in Kargar and An (2015), as the outer layer of our keyword search engine.

Algorithm 3 Lawler-based retrieval algorithm

Input: graph G, query Q, and k
Output: set of top-k ordered approx-MCC_rs

```
1  for i←1 to l do
2      C_i←set of nodes in G containing k_i
3  C←<C_1, C_2, …, C_l>
4  Queue←0
5  A←BKSR_A(C, G, l)
6  if A≠∅ then
7      insert <A, C> into Queue
```

```
8  while Queue≠∅ do
9    <A, S>←top element of Queue
10   output(A)
11   k←k−1
12   if k=0 then
13     return
14   <SB1, SB2, …, SBl>←subspaces(A, S)
15   for i←1 to l do
16     Ai←BKSRA(SBi, G, l)
17     if Ai≠∅ then
18       insert <Ai, SBi> into Queue
```

Proc subspaces (Kargar and An, 2015)

Input: the best answer of the previous step, $A=<v_1, v_2, \ldots, v_l>$, and search space C
Output: l new subspaces

```
1  for i←1 to l do
2    for j←1 to i−1 do
3      SBi^j←{vj}
4    SBi^j←Ci−{vi}
5    for j←i+1 to l do
6      SBi^j←Cj
7  return <SB1, SB2, …, SBl> where SBi=<SBi^1, SBi^2, …, SBi^l> represents space SBi^1×SBi^2×…×SBi^l.
```

The procedure of finding the best answer recurs ($k\times l-l+1$) times in the Lawler-based top-k answer retrieval. Therefore, its performance can significantly affect the overall performance of the algorithm. In this work, we propose algorithm BKSR(A), which is an approximate version of algorithm BKSR, to find the best approximate result of a subspace. Algorithm BKSR(A) receives a search space as $C=<C_1, C_2, \ldots, C_l>$ and sorts it based on the size of the subsets. Then, for each vertex of the smallest size subset, the set P is set equal to the sum of the other subsets and the set R is set to empty. The vertices of the smallest size subset are considered as the roots of recursive trees. The recursive trees are expanded by selecting a vertex of each level that has the least distance to the vertices of R. The only MCC_r retrieved from each recursive tree is added to the priority queue based on its weight. The minimum weight element of the priority queue is then returned as the best result. The pseudo code is shown in Algorithm 4.

Algorithm 4 BKSR(A)

Input: search space C, graph G, and number of keywords l
Output: the best approximate MCC_r in C

```
1  Queue←{}
2  for i←1 to l do
3    orderi←Ci
4  end for
5  sort orderi in ascending order of size
6  for i←2 to l do
7    P←P∪orderi
8  end for
9  for j←1 to size(order1) do
10   i←1
11   R←{order1^j}
12   P←P∩Γ(order1^j)
13   BKSW(P, R, i)
14 end for
15 return top element of Queue
```

Proc BKSW(P, R, i)

```
1  if P={} then
2    insert R into Queue
3  end if
4  i←i+1
5  choose a pivot u∈Ci∩P
   // choose u to maximize |(Ci∩P)∩Γ(u)|
6  S←(Ci∩P)\Γ(u)
7  v←S^1
8  for i←2 to size(S) do
9    for j←1 to size(R)
10     if dist(S^i, R^j)<dist(v, R^j) then
11       v←S^i
12     end if
13   end for
14   if size((P∩Γ(v))∪(R∪{v}))≥l then
15     BKSW(P∩Γ(v), R∪{v}, i)
16   end if
17 end for
18 return
```

In algorithm BKSR(A), order_l^j shows the j^{th} vertex of the l^{th} set of order. Similarly, in the other expressions, the superscript is related to the vertices and the subscript is related to the sets.

Theorem 5 Procedure BKSR(A) produces an answer with two-approximation.

Proof The answers of BKSR(A) are always MCC_rs in which the distance between every two vertices is no larger than r. Therefore, the weight of any MCC_r retrieved by BKSR(A) satisfies the following equation:

$$\mathrm{approx\_weight}(\mathrm{MCC}_r) \le l(l-1)r/2. \quad (5)$$

Algorithm r-clique generates answers with two-approximation. The answers generated by this algorithm are star-shaped trees with a center node that

directly connects to (l−1) leaf nodes. Therefore, the maximum weight of answers retrieved by algorithm r-clique is

$$\begin{aligned}\max\_\text{weight}(r_{\text{clique}}) &= (l-1)r + 2r\left[\frac{l(l-1)}{2} - (l-1)\right] \\ &= (l-1)^2 r.\end{aligned} \tag{6}$$

In Kargar and An (2015), it was proved that the weight of answers retrieved by r-clique is at most twice the optimal weight. Therefore, it can be concluded that

$$\max\_\text{weight}(r_{\text{clique}}) \le 2\,\text{opt}\_\text{weight}(r_{\text{clique}}). \tag{7}$$

It can be simply proved that the optimal r-clique is always an MCC_r and that this MCC_r is the optimal one among all the relevant MCC_rs to the query. It means that

$$\text{opt}\_\text{weight}(r_{\text{clique}}) = \text{opt}\_\text{weight}(\text{MCC}_r). \tag{8}$$

Considering Eqs. (5)–(7), it is concluded that

$$\begin{aligned}\text{approx}\_\text{weight}(\text{MCC}_r) &\le \frac{l(l-1)r}{2} \le (l-1)^2 r \\ &\le 2\text{opt}\_\text{weight}(r_{\text{clique}}).\end{aligned} \tag{9}$$

According to Eq. (8) and inequality (9), we have

$$\text{approx}\_\text{weight}(\text{MCC}_r) \le 2\,\text{opt}\_\text{weight}(\text{MCC}_r). \tag{10}$$

It shows that the weight of answers retrieved by BKSR(A) is lower than twice that of the optimal MCC_r.

Theorem 6 The time complexity of algorithm BKSR(A) in the worst case is $O(l^2|C_{\min}||C_{\max}|)$.

Proof Suppose the smallest size set is shown by $C_{\min}$. In algorithm BKSR(A), the procedure BKSW is called $|C_{\min}|$ times. This procedure is recursive and its recursive tree is similar to a chain containing at most l recalls of BKSW. In any of recursive calls of BKSW, the intersection operations need $O(|C_{\max}|)$ for calculation. The statement 8 repeats at most $|C_{\max}|l$ times. Each time, the candidate vertex is compared with a vertex of R (the current clique), while the size of R is at most l. Therefore, the time complexity of BKSW, in the worst case, is equal to $O(|C_{\max}|)l^2+|C_{\max}|l= O(|C_{\max}|l^2)$. In BKSR($A$), procedure BKSW is called for $|C_{\min}|$ vertices. Therefore, the time complexity of BKSR(A) is $O(l^2|C_{\min}||C_{\max}|)$.

Kargar and An (2011, 2015) also proposed a greedy algorithm to approximate r-cliques. This algorithm estimates results in two-approximate order. It means that the weight of each result generated is no higher than twice the weight of the optimal results. However, results produced by this algorithm are not essentially r-clique and can have the radius of $2r$. The time complexity of this algorithm is $O(l^2|C_{\max}|^2)$. In contrast, BKSR(A) produces results with the maximum radius r. Therefore, it is more accurate than r-clique. On the other hand, in the time complexity calculations of BKSR, we assume that all of a group's keyword nodes are tested in each recursion call, while in practice, only the set of vertices that are connected to the selected ones in the earlier levels are examined at each level. Thus, the runtime of BKSR(A) is in practice lower than that of r-clique. The experimental comparisons of these algorithms are presented in Section 5.

5 Experiments

To evaluate the effectiveness and efficiency of the proposed algorithms, we conduct extensive experiments on the two large-scale real-world datasets IMDb (http://www.grouplens.org/node/73) and DBLP (http://dblp.uni-trier.de/xml/). A corresponding graph is extracted from each of the datasets by considering any tuple as a vertex and any connection between two tuples through foreign keys as an edge of the graph. Similar to Ding et al. (2007), Qin et al. (2009), and Kargar and An (2015), any edge between vertices v_i and v_j is weighted based on Eq. (11) such that $\deg(v_k)$ shows the degree of vertex v_k:

$$w(v_i, v_j) = \frac{1}{2}\left[\log(1+\deg(v_i)) + \log(1+\deg(v_j))\right]. \tag{11}$$

We also use uniform weighting as an alternative way to determine the weights of graph's edges, which gives each edge a weight of one.

In this section, we evaluate the results retrieved by approximate versions of BKS, BKSR, BKSM, and BKSRM algorithms, which are named BKS(A),

BKSR(A), BKSM(A), and BKSRM(A), respectively. Any of the approximate algorithms is extracted from its corresponding exact algorithm by adapting them to Algorithm 4. The results are presented in comparison with the results of r-clique and r-clique-rare algorithms (Kargar and An, 2015). The reasons why we choose r-clique algorithms for comparison are that: (1) they are among the most recent work in the keyword search domain; (2) the definition of r-cliques (as the results) is more similar to that of the MCC_r; (3) they produce top-k answers in a polynomial delay, similar to our algorithms.

All of the algorithms are implemented using Java and on an Intel® Core™ i7@2.80 GHz laptop with 16 GB RAM and the Windows 8 operating system. PostgreSQL is used to store the relational data. The different systems are compared in terms of effectiveness and efficiency. To keep the comparison fair, all the experiments are executed on the same set of queries as in Kargar and An (2015).

5.1 Datasets and queries

In the experiments, the two large-scale real-world datasets IMDb and DBLP are employed to evaluate the effectiveness and efficiency of the proposed algorithms. The IMDb dataset shows the relationships among the users and the movies of website IMDb and the rating of the users to the movies. The numbers of tuples in three relations (user, movie, and rating) in the IMDb dataset are 138 493, 131 262, and 20 000 263, respectively. The DBLP dataset contains information about papers, authors, and paper citations. The numbers of tuples in author, paper, authorship, and citations are 613 000, 929 000, 2 375 000, and 82 000, respectively. The set of queries and their frequencies in the graphs of DBLP and IMDb are presented in Table 2. Note that the queries used for testing are the same as the queries used in Qin et al. (2009) and Kargar and An (2011, 2015) for better evaluation and comparison.

Table 2 The queries on DBLP and IMDb datasets

Frequency	DBLP query	IMDb query
0.0003	Q_1: distance, discovery, scalable, protocols	Q_6: game, summer, bride, dream
0.0006	Q_2: Graph, routing, space, scheme	Q_7: Friday, street, party, heaven
0.0009	Q_3: fuzzy, optimization, development, support, environment, database	Q_8: girl, lost, blood, star, death, all
0.0012	Q_4: modeling, logic, dynamic, application	Q_9: city, world, blue, American
0.0015	Q_5: control, web, parallel, algorithms	Q_{10}: king, house, night, story

5.2 Efficiency evaluation

In this subsection, the runtime of different systems is compared based on the performance affecting factors including the value of r (radius of results), the number of keywords (l), and the frequency of keywords (f_{k_i}). In the diagrams, the fixed factors are set to r=11, l=4, $f_{k_i}=0.0009$ for all the queried keywords. The average execution times of the systems when varying the radius of results (r) are shown in Fig. 4a, showing that the execution times of different systems are increased by increasing the value of r. This is because of the greater number of edges which should be examined when increasing the radius of results. Fig. 4 shows that algorithms BKS(A) and BKSR(A) are faster than r-clique and that the execution time of BKSR(A) is close to that of r-clique-rare, while as we will show later, the accuracy of BKSR(A) is higher than that of r-clique-rare. The same behavior is maintained in Figs. 4b and 4c, showing that the execution times grow exponentially with an increased number of queried keywords or querying the more

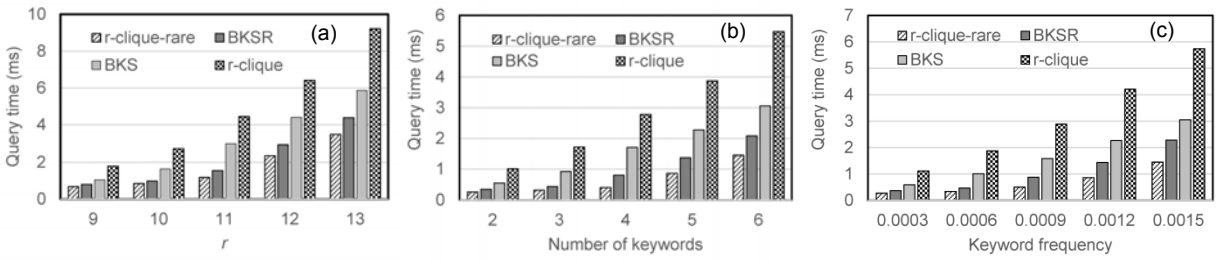

Fig. 4 Execution times of the systems based on different factors over the IMDb dataset: (a) query time vs. r; (b) query time vs. the number of keywords; (c) query time vs. the keyword frequency

frequent keywords. The BKS(A) and BKSR(A) systems over the IMDb dataset are approximately two and four times faster than r-clique, respectively.

As mentioned before, one of the advantages of the proposed algorithms is their ability for distributive execution with regard to the isolation building of the recursive trees. The approximate versions of the distributed algorithms BKSM and BKSRM, named BKSM(A) and BKSRM(A) respectively, have been executed with eight simultaneous processors in a distributed shared memory system. The runtimes of these algorithms for different frequencies of keywords are shown in Fig. 5a. The outperformance of BKSRM(A) over BKSM(A) is evident. Comparing Fig. 5a with the similar diagram in Fig. 4 shows that the execution times of BKSM(A) and BKSRM(A) are much lower than those of the r-clique and r-clique-rare systems. Note that r-clique-based systems cannot be executed in a distributive manner because any of their results are formed by considering all of the keyword nodes in the graph. We also show the execution times of BKSM(A) and BKSRM(A) systems when varying the number of processors over the queries with an average frequency 0.0015 in Fig. 5b. As expected, by increasing the number of processors, the execution time of distributed systems decreases due to the increase of degree of parallelism. However, managing multiple processors imposes an overhead, which causes a slowing of the performance improvement gained by the large number of processors.

The same experiments are conducted on the DBLP dataset (Fig. 6). Because of the greater number of relationships in the DBLP dataset than in the IMDb, the average time to respond to the DBLP queries is on average higher than that to IMDb queries. Fig. 6a shows that the execution times of all the examined systems increase with the increase of the radius of results. Fig. 6a also shows that algorithms BKS(A) and BKSR(A) are faster than r-clique and that the execution time of BKSR(A) is close to that of r-clique-rare. This behavior is also maintained when varying the number of queried keywords and the frequency of keywords.

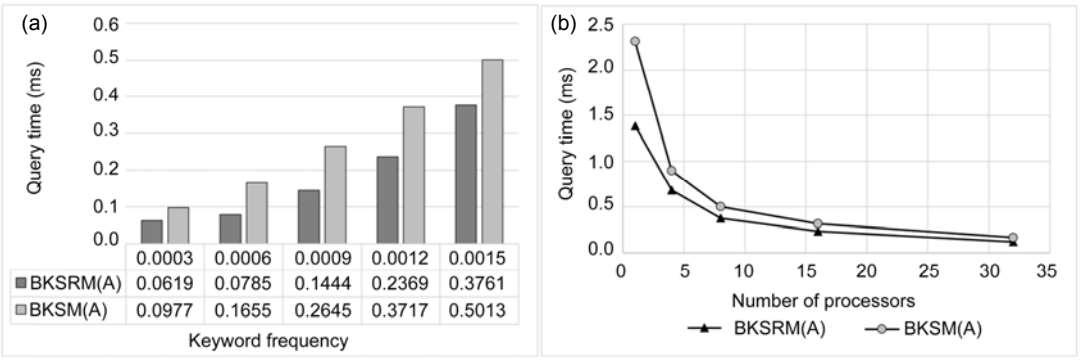

Fig. 5 The time of executing the distributed systems with eight processors over the IMDb dataset (a) and the execution time of distributed systems when varying the number of processors (b)

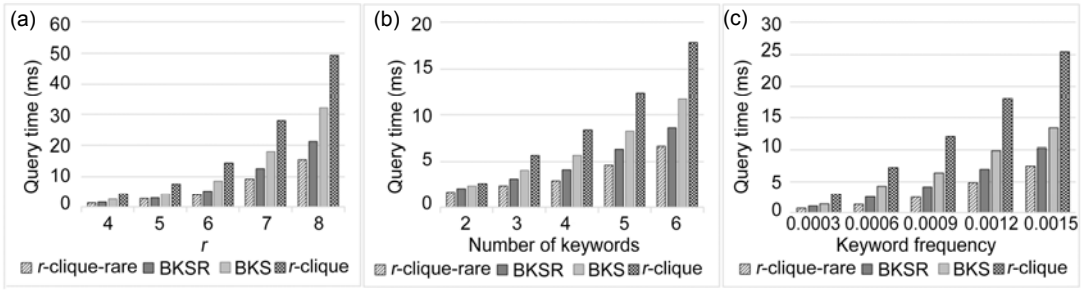

Fig. 6 Execution times of different algorithms over the DBLP dataset: (a) query time vs. r; (b) query time vs. the number of keywords; (c) query time vs. the keyword frequency

5.3 Effectiveness evaluation based on the compactness measure

As we mentioned before, the proposed algorithms can generate results in which no distances are larger than r. However, the radius of results generated by r-clique and r-clique-rare algorithms may reach $2r$. Fig. 7a shows the average percentage of cliques with the maximum distance r on the IMDb dataset using algorithm BKS(A) as an envoy of all of the other proposed algorithms. The diagram is plotted based on the top 10, 20, 30, 40, and 50 results of the examined queries.

Since BKS(A) generates MCC$_r$s with maximum distance r, its average percentage of generating such results is always 100%. According to Fig. 7a, for the top-10 retrieved results, 90% and 58% of results generated by r-clique and r-clique-rare systems respectively are cliques with maximum distance r. This shows that the r-clique system can generate more high-quality results than the r-clique-rare. However, by increasing the number of top results, the ability of both algorithms in generating the compact results is reduced such that when retrieving the top-50 results for each query, their percentages of generating desired results are 62.12% and 31.5%, respectively. Fig. 7b is plotted based on the different frequencies of keywords. This diagram also shows that the compactness of results decreases in the r-clique and r-clique-rare algorithms by increasing the frequency of keywords. It is because of the greater number of candidate results when querying high-frequency keywords. In contrast, the results retrieved by the proposed system all satisfy the maximum distance r independent of the frequency of keywords.

5.4 Effectiveness evaluation based on the average weight measure

To make quality measurement possible, all the results of a keyword query are detected using the exact BKSR and among them, top-k minimum weight results are selected as the ground-truth results of the query. For example, for the keywords with frequency 0.0003, there are 3 043 836 results in the IMDb dataset, and retrieving them takes two hours and forty minutes. Among them, top-50 results are selected and used as the "Exact" ones to measure the quality of results generated by different systems. In the above example, the average uniform weight of the selected top-50 results is six and their average logarithmic weight is 38.1731. Generating top-50 results by approximate algorithms BKSRM(A), BKSM(A), r-clique-rare, BKSR(A), BKS(A), and r-clique systems takes 0.062, 0.097, 0.287, 0.377, 0.6, and 1.117 ms, respectively. Note that the uniform weight of each answer is determined by Eq. (1) when the weight of each edge is set to one. Similarly, the logarithmic weight of an answer is determined by Eq. (1) when the weight of each edge is calculated by Eq. (11).

Figs. 8a and 8b show the average uniform weight of results retrieved by the approximate algorithms over the IMDb dataset when the frequencies of keywords are 0.0003 and 0.0015, respectively. As is evident, the uniform weights of results generated by BKS(A) and BKSR(A) algorithms are equal to the weight of the exact results, while r-clique and r-clique-rare produce results with higher uniform weights. In addition, the uniform weights of r-clique are closer to the exact weights rather than those of r-clique-rare. Fig. 8b shows that, in the best and

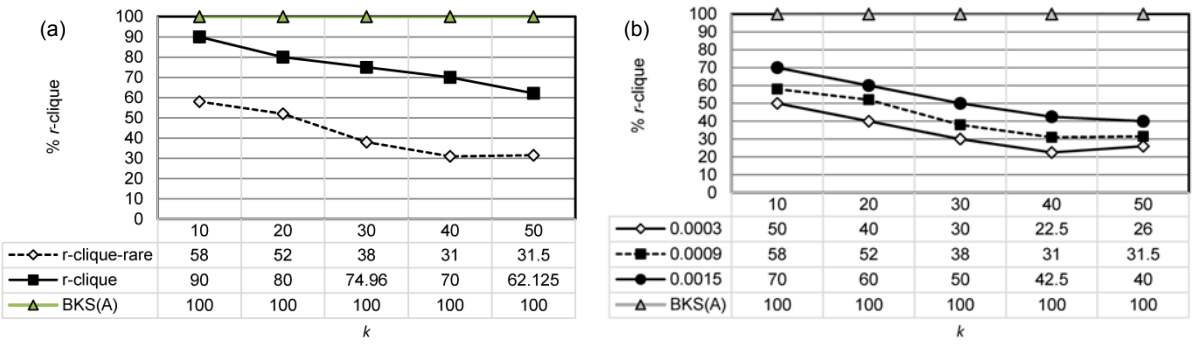

Fig. 7 The average percentage of generating results of the maximum distance r by different systems on the IMDb dataset when varying the value of k (a) and the average percentage of generating results of the maximum distance r when varying the frequency of keywords (b)

worst cases, the *r*-clique system produces results weighted 1.67% and 6.67% higher than the exact weights, respectively. These values are 10% and 26.67% for the results of the *r*-clique-rare system.

Fig. 9a shows the average logarithmic weights of results retrieved by the different systems on the IMDb dataset when the frequency of keywords is 0.0003. The closeness of the weight of the results produced by the proposed systems to the exact weights indicates the success of these systems in retrieving high-quality results. However, as expected, the logarithmic weights of results produced by *r*-clique and *r*-clique-rare are very much higher than the exact weights. Fig. 9b, which is plotted based on the more frequent keywords, shows the close performance of BKS(*A*), BKSR(*A*), and *r*-clique in terms of the average logarithmic weight of results. However, as shown previously, BKS(*A*) and BKSR(*A*) are significantly faster than *r*-clique in retrieving results.

6 Conclusions

In this paper, we have proposed a group of scalable branch and bound algorithms which retrieve MCC_r as the results of a keyword query. These algorithms employ different strategies of pruning to reduce the time of answering a keyword query. In addition, the isolated recursive recalls of different roots provide the ability for parallel processing of these algorithms. The time complexity of the proposed algorithms shows, in theory, the performance of the proposed algorithms in comparison to the previous works. This efficiency is proved in practice by experiments on two real-words datasets. The experimental results showed that algorithms BKS(*A*) and BKSR(*A*) are two and four times faster than the *r*-clique system, respectively, which is also a clique-based retrieval system. In addition, algorithms BKSM(*A*) and BKSRM(*A*) with the ability of parallel

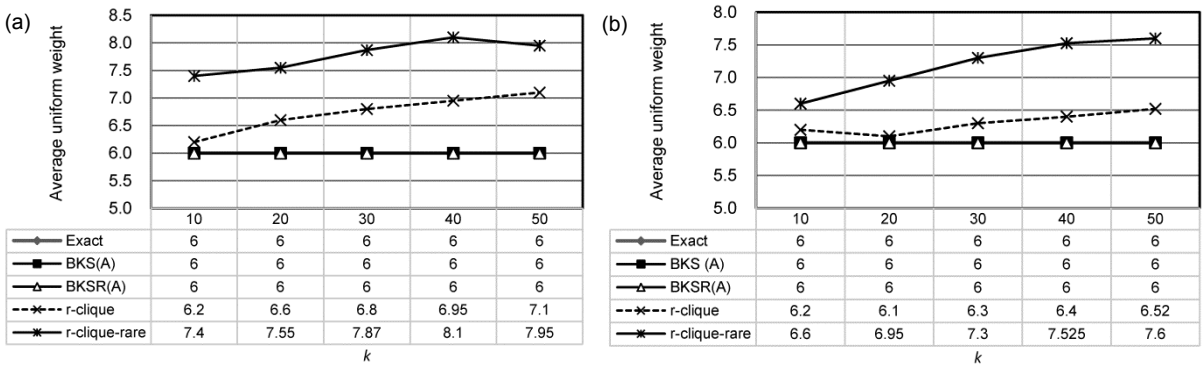

Fig. 8 The average uniform weights of results when querying the keywords with frequency 0.0003 (a) and 0.0015 (b)

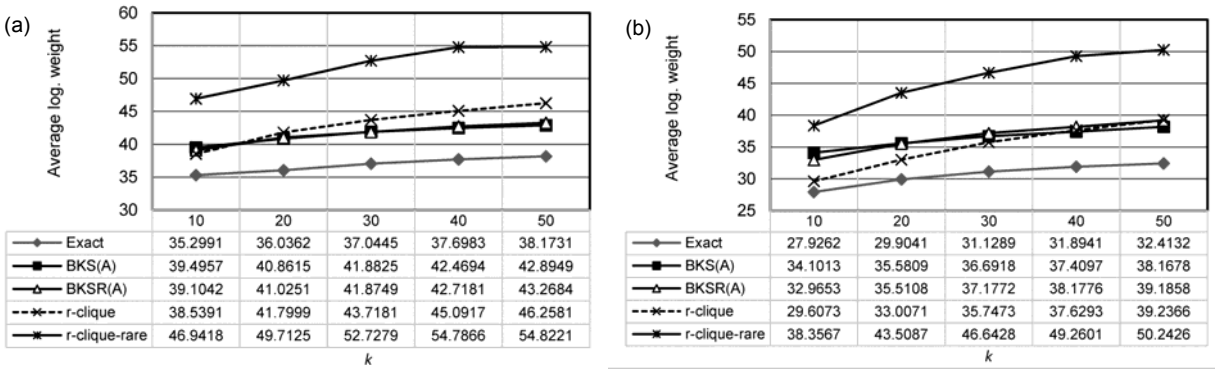

Fig. 9 The average logarithmic weight of results when querying the keywords with frequency 0.0003 (a) and 0.0015 (b)

processing are more efficient than the *r*-clique and *r*-clique-rare systems. In the effectiveness evaluation based on a compactness measure, the proposed algorithms significantly outperformed those in the previous works in terms of generating results with maximum distance *r*. This is because of observing the maximum radius *r* in generating all of the results by the proposed systems. The effectiveness of these algorithms was also confirmed by examining the average weights of results.

Contributors

Asieh GHANBARPOUR completed the proofs and mathematical parts, drafted the manuscript, and revised it. Khashayar NIKNAFS completed the algorithms, carried out the implementations, and evaluated the results. Hassan NADERI guided the research and supervised the writing of the manuscript.

Compliance with ethics guidelines

Asieh GHANBARPOUR, Khashayar NIKNAFS, and Hassan NADERI declare that they have no conflict of interest.

References

Bergamaschi S, Guerra F, Interlandi M, et al., 2013. QUEST: a keyword search system for relational data based on semantic and machine learning techniques. *Proc VLDB Endowm*, 6(12):1222-1225. https://doi.org/10.14778/2536274.2536281

Bergamaschi S, Guerra F, Interlandi M, et al., 2016. Combining user and database perspective for solving keyword queries over relational databases. *Inform Syst*, 55:1-19. https://doi.org/10.1016/j.is.2015.07.005

Bron C, Kerbosch J, 1973. Finding all cliques of an undirected graph. *Commun ACM*, 16(9):575-577. https://doi.org/10.1145/362342.362367

Calado P, da Silva AS, Laender AHF, et al., 2004. A Bayesian network approach to searching Web databases through keyword-based queries. *Inform Process Manag*, 40(5): 773-790. https://doi.org/10.1016/j.ipm.2004.03.002

Dasari NS, Ranjan D, Mohammad Z, 2014. Maximal clique enumeration for large graphs on Hadoop framework. Proc 1st Workshop on Parallel Programming for Analytics Applications, p.21-30. https://doi.org/10.1145/2567634.2567640

de Virgilio R, Cappellari P, Miscione M, 2009. Cluster-based exploration for effective keyword search over semantic datasets. In: Laender AHF, Castano S, Dayal U, et al. (Eds.), Conceptual Modeling - ER 2009. Springer Berlin Heidelberg, p.205-218. https://doi.org/10.1007/978-3-642-04840-1_17

Ding BL,Yu JX, Wang S, et al., 2007. Finding top-*k* min-cost connected trees in databases. IEEE 23rd Int Conf on Data Engineering, p.836-845. https://doi.org/10.1109/ICDE.2007.367929

Ghanbarpour A, Naderi H, 2018. A model-based keyword search approach for detecting top-*k* effective answers. *Comput J*, 62(3):377-393. https://doi.org/10.1093/comjnl/bxy056

Golenberg K, Kimelfeld B, Sagiv Y, 2008. Keyword proximity search in complex data graphs. Proc ACM SIGMOD Int Conf on Management of Data, p.927-940. https://doi.org/10.1145/1376616.1376708

Guo L, Shao F, Botev C, et al., 2003. XRANK: ranked keyword search over XML documents. Proc ACM SIGMOD Int Conf on Management of Data, p.16-27. https://doi.org/10.1145/872757.872762

Hao YF, Cao HP, Qi Y, et al., 2015. Efficient keyword search on graphs using MapReduce. IEEE Int Conf on Big Data, p.2871-2873. https://doi.org/10.1109/BigData.2015.7364106

He H, Wang HX, Yang J, et al., 2007. BLINKS: ranked keyword searches on graphs. Proc ACM SIGMOD Int Conf on Management of Data, p.305-316. https://doi.org/10.1145/1247480.1247516

Kargar M, An AJ, 2011. Keyword search in graphs: finding *r*-cliques. *Proc VLDB Endowm*, 4(10):681-692. https://doi.org/10.14778/2021017.2021025

Kargar M, An AJ, 2015. Finding top-*k* *r*-cliques for keyword search from graphs in polynomial delay. *Knowl Inform Syst*, 43(2):249-280. https://doi.org/10.1007/s10115-014-0736-0

Kargar M, An AJ, Yu XH, 2014. Efficient duplication free and minimal keyword search in graphs. *IEEE Trans Knowl Data Eng*, 26(7):1657-1669. https://doi.org/10.1109/TKDE.2013.85

Kim S, Lee W, Arora NR, et al., 2012. Retrieving keyworded subgraphs with graph ranking score. *Expert Syst Appl*, 39(5):4647-4656. https://doi.org/10.1016/j.eswa.2011.08.136

Kimelfeld B, Sagiv Y, 2006. Finding and approximating top-*k* answers in keyword proximity search. Proc 25th ACM SIGMOD-SIGACT-SIGART Symp on Principles of Database Systems, p.173-182. https://doi.org/10.1145/1142351.1142377

Lawler EL, 1972. A procedure for computing the *K* best solutions to discrete optimization problems and its application to the shortest path problem. *Manag Sci*, 18(7):401-405. https://doi.org/10.1287/mnsc.18.7.401

Le TN, Bao FZ, Ling TW, 2015. Exploiting semantics for XML keyword search. *Data Knowl Eng*, 99:105-125. https://doi.org/10.1016/j.datak.2015.06.003

Li GL, Ooi BC, Feng JH, et al., 2008. EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. Proc ACM SIGMOD Int Conf on Management of Data, p.903-914. https://doi.org/10.1145/1376616.1376706

Liu F, Yu C, Meng WY, et al., 2006. Effective keyword search in relational databases. Proc ACM SIGMOD Int Conf on Management of Data, p.563-574.

https://doi.org/10.1145/1142473.1142536
Mass Y, Sagiv Y, 2012. Language models for keyword search over data graphs. Proc 5th ACM Int Conf on Web Search and Data Mining, p.363-372. https://doi.org/10.1145/2124295.2124340
Mesquita F, da Silva AS, de Moura ES, et al., 2007. LABRADOR: efficiently publishing relational databases on the web by using keyword-based query interfaces. *Inform Process Manag*, 43(4):983-1004. https://doi.org/10.1016/j.ipm.2006.09.018
Nguyen K, Cao JL, 2012. Top-*K* data source selection for keyword queries over multiple XML data sources. *J Inform Sci*, 38(2):156-175. https://doi.org/10.1177/0165551511435875
Ning XM, Jin H, Jia WJ, et al., 2009. Practical and effective IR-style keyword search over semantic web. *Inform Process Manag*, 45(2):263-271. https://doi.org/10.1016/j.ipm.2008.12.005
Pan YF, Wu YQ, 2013. ROU: advanced keyword search on graph. Proc 22nd ACM Int Conf on Information & Knowledge Management, p.1625-1630. https://doi.org/10.1145/2505515.2505743
Park CS, Lim S, 2015. Efficient processing of keyword queries over graph databases for finding effective answers. *Inform Process Manag*, 51(1):42-57. https://doi.org/10.1016/j.ipm.2014.08.002
Park J, Lee SG, 2011. Keyword search in relational databases. *Knowl Inform Syst*, 26(2):175-193. https://doi.org/10.1007/s10115-010-0284-1
Qin L, Yu JX, Chang LJ, et al., 2009. Querying communities in relational databases. Proc IEEE 25th Int Conf on Data Engineering, p.724-735. https://doi.org/10.1109/ICDE.2009.67
Sagharichian M, Naderi H, Haghjoo M, 2015. ExPregel: a new computational model for large-scale graph processing. *Concurr Comput Pract Exp*, 27(17):4954-4969. https://doi.org/10.1002/cpe.3482
Segundo PS, Lopez A, Batsyn M, et al., 2016. Improved initial vertex ordering for exact maximum clique search. *Appl Intell*, 45(3):868-880. https://doi.org/10.1007/s10489-016-0796-9
Segundo PS, Artieda J, Strash D, 2018. Efficiently enumerating all maximal cliques with bit-parallelism. *Comput Oper Res*, 92:37-46. https://doi.org/10.1016/j.cor.2017.12.006
Tomita E, Tanaka A, Takahashi H, 2006. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theor Comput Sci*, 363(1):28-42. https://doi.org/10.1016/j.tcs.2006.06.015
Wang D, Zou L, Zhao DY, 2015. Top-*k* queries on RDF graphs. *Inform Sci*, 316:201-217. https://doi.org/10.1016/j.ins.2015.04.032
Xu YW, Guan JH, Li FR, et al., 2013. Scalable continual top-*k* keyword search in relational databases. *Data Knowl Eng*, 86:206-223. https://doi.org/10.1016/j.datak.2013.03.004
Yu T, Liu MC, 2017. A linear time algorithm for maximal clique enumeration in large sparse graphs. *Inform Process Lett*, 125:35-40. https://doi.org/10.1016/j.ipl.2017.05.005
Yuan Y, Wang GR, Chen L, et al., 2013. Efficient keyword search on uncertain graph data. *IEEE Trans Knowl Data Eng*, 25(12):2767-2779. https://doi.org/10.1109/TKDE.2012.222

Appendix: Proof of Theorem 2

Let $R=\{v_{q1}, v_{q2}, \ldots, v_{qd}\}$ show the set of all the vertices that have been added to the MCC$_r$ so far. In this situation, the set of P contains

$$g=V\cap\Gamma(v_{q1})\cap\Gamma(v_{q2})\cap\ldots\cap\Gamma(v_{qd}),$$

where in the initial state, $g=V$ and $R=\varnothing$.

Assume that $X_i=R_i\cup P_i$ shows the set of all the vertices that are involved in the MCC$_r$ construction, and suppose that all the MCC$_r$s containing $Q\cup\{q'\}$ have been retrieved without duplication for any $q'\in R$. Consider vertex u is selected as a pivot in statement 5 of procedure BKS. According to this selection, the set of nodes that can be added to the subgraph would be

$$\text{Cand}_i=(C_i\cap P)\backslash\Gamma(u)=\{v_1, v_2, \ldots, v_i\}.$$

At statement 6, vertex v_j is chosen from Cand$_i$ to expand the subgraph. Accordingly, sets P and R would be updated as follows for the next call:

$$P_{i+1}=P_i\cap\Gamma(v_j),\ R_{i+1}=R_i\cup\{v_j\}.$$

We claim that: (1) by expanding every $v_j\in\text{Cand}_i$, all non-duplicate MCC$_r$s containing $g\cup\{v_j\}$ will be generated; (2) when finishing the generation of MCC$_r$s following this expansion, there is no new MCC$_r$ containing $g\cup\{v_j\}$.

We prove our claims by induction on $|X_i|=n$.

When $1\le n\le 2$, the satisfying of the claim is easily confirmed.

Now, we assume that the claim is true for all $n\le N$ and prove that this claim is true for $n=N+1$.

To this end, consider step i of the recursion when working with Cand$_i$ as the candidate vertices to expand the subgraph in step i, R_i as the set of selected vertices so far, and P_i as the set of all the vertices that can be added to the uncompleted MCC$_r$. By assuming that the candidate vertices are selected in order, first

we examine the expansion of the subgraph by vertex v_1. In this case, two states may occur:

1. If $P_i=\varnothing$, then the set of vertices $R_i\cup\{v_1\}$ is considered as an MCC$_r$. The current recursion of BKS has been executed just because $|R_i\cup\{v_1\}|\geq l$ is satisfied in the previous step of recursion. In other words, if this inequality is not satisfied in step $i-1$, the algorithm will never enter step i. On the other hand, since there are at most l different types of vertices in the graph, the height of recursion is not larger than l. Therefore, in this case, the size of $R_i\cup\{v_1\}$ is exactly equal to l. Since there is not an edge among the vertices of each type, the vertices of the MCC$_r$ involved are from different types. Consequently, no subgraph of the MCC$_r$ can cover the query. It shows that $R_i\cup\{v_1\}$ is a minimal r-clique covering l keywords of the query.

2. If $P_i\neq\varnothing$, then the stages of pivoting and recursion would be followed. Let in this step vertex $u'\in C_i\cap P_i$ be selected as the pivot. The following two states may occur in processing the loop:

(1) If $(C_i\cap P_i)\backslash\Gamma(u')\neq\varnothing$, then one vertex (e.g., v_k) of this set is selected to be added to the subgraph and the BKS procedure is recalled if statement 7 is satisfied. Executing this procedure leads to producing all MCC$_r$s containing $R_i\cup\{v_1\}$ and without duplication because $|X_{i+1}|=|P_{i+1}+R_{i+1}|\leq N$ and the induction hypothesis holds true for it.

(2) If $(C_i\cap P_i)\backslash\Gamma(u')=\varnothing$, then the recall of the BKS procedure would not proceed and no subgraph containing $R_i\cup\{v_1\}$ would be presented as MCC$_r$. More precisely, in this case, there is no new vertex to expand the subgraph while the depth of recursion does not reach l. It means that the uncompleted subgraph does not cover all the query keywords and it is not truly an MCC$_r$.

After the examination of vertex v_1, if $(C_i\cap P_i)\backslash\Gamma(u')-\{v_1\}=\varnothing$, its other vertices are selected step by step and for them, the above discussions follow.

In what follows, we discuss the completeness of the result set generated by the proposed algorithms. Since any MCC$_r$ should contain a vertex of every type, and all its vertices are directly connected, the order of applying type restrictions would not be affected in the final set of MCC$_r$s. However, there may be a covered r-clique $g'\subseteq G(V, E\cup E')$, where E' is the set of deleted edges that connect the vertices with the same type and g' contains some edges $e'\subseteq E'$. If g' is retrieved by BKS, it certainly covers all the queried keywords. However, any edge of e' shows the existence of two vertices with the same keyword in g'. Therefore, subgraph $g''=g'-e'$ also covers the query, and this implies that g' is not a minimal covered clique.

The above discussions show that the claim holds for any positive integer n as a result of induction.