ROU: Advanced Keyword Search on Graph

Yifan Pan
Indiana University
panyif@cs.indiana.edu

Yuqing Wu
Indiana University
yuqwu@cs.indiana.edu

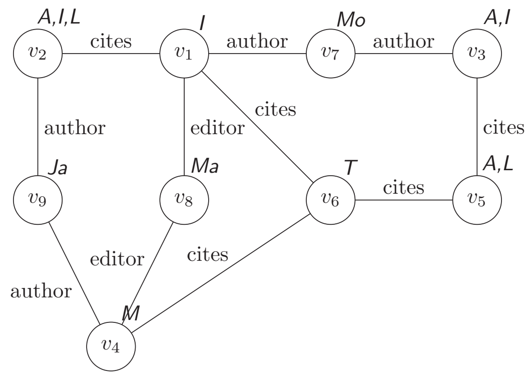
ABSTRACT

Keyword search, due to its simplicity in expressing search demands, has been the major means of search queries for Internet search engines. More recently, the notion has been explored in structured and semi-structured data, where new search semantics were defined, and evaluation algorithms proposed. What is yet to be explored thoroughly in keyword search on those types of graph data is how optional and negative keywords can be expressed, what the results should be and how such search queries can be evaluated efficiently. In this paper, we propose and formally define a new type of keyword search query, ROU-query, which takes as input keywords in three categories: required, optional and unwanted, and returns as output sets of data entries (nodes in the data graph) whose neighborhood satisfy the keyword requirements. We define multiple semantics, including maximal coverage and minimal footprint, to ensure the meaningfulness of the results. We propose a new data structure, query induced partite graph (QuIP), that can capture the constraints related to neighborhood size and unwanted keywords, and propose a family of algorithms that take advantage of the information in QuIP for efficient evaluation of ROU-queries. We conducted extensive experimental evaluations to analyze the performance of our proposed algorithms and found that the size of QuIP is generally very small compared to the data graph, and our algorithms are able to effectively prune non-promising partial results and generate results for ROU-queries efficiently.

1. INTRODUCTION

Keyword search has been proven as an effective method for information retrieval, most notably used in search engines such as Google and Bing. While traditionally keyword search has been focusing on finding particular entities (documents, webpages, images, videos, etc.) [13], recently we witnessed significant efforts towards applying keyword search to structured and semi-structured data, including relational data [1,9,12], XML data [3] or general graph [4,7,8], to find sets of data entries, in the form of Steiner trees [1,4,5,7,8] and connected sub-graphs [10,11,14], that satisfy keyword and connectivity constraints.

Some of the earlier works focused on returning (approximate) minimum Group Steiner Trees (GST) that connect keyword nodes [1,4,5,7,8], featuring different GST generation approaches and various scoring functions for ranking the resultant GSTs. More recently, the focus has been shifted from ranking results to finding clusters of GSTs that deliver richer and more cohesive information, rendering the results beyond trees, into sub-graphs, such as r-radius Steiner graphs [11], keyword community [14] and r-cliques [10]. The common themes of most of these works are: (1) one set of keywords are given as input and the AND semantics is enforced, i.e. all of which are required to appear in each result; and (2) scoring functions based on node and edge weights are used to identify the top-k results that are considered most relevant to the keyword set.



Node	Content
v_1	**_I_ntelligence**: Can Machines Beat Humans?
v_2	**_A_rtificial _I_ntelligence** and Human **_L_anguage**
v_3	**_A_rtificial _I_ntelligence** Handbook
v_4	Secrets of Human **_M_ind**
v_5	Natural **_L_anguage** and **_A_rtificial** Life
v_6	Machine **_T_ranslation**
v_7	Jouke **_Mo_lenaar**
v_8	Fabian **_Ma_rtinez**
v_9	Alexander **_Ja_cobsen**

Figure 1: Example Graph

[7] and [11], while still taking one keyword set as input, employed the OR semantics, allowing results to contain *some* (but not necessarily *all*) keywords. However, due to this relaxation, they lost the ability to demand certain keywords *must* appear in every result. Additional works on

such relaxed semantics including [9] and [12] on relational databases and [3] on XML documents. However, they all assumed the existence of schema information, which is not applicable to general schema-free graph data.

Meanwhile, negative predicate, while a key component in Boolean queries [13], has not been studied in the landscape of keyword search queries on graphs.

The issues listed above strictly limit what a traditional keyword search query can express. For example, consider the example graph (with the graph structure shown in Figure 1 and the contents of the nodes shown in the table), which is a fragment of the citation network. All the content words can potentially be keywords for different queries but keywords that appear in our example queries throughout the paper are in **Bold** and the abbreviation of these keywords marked in the figure. It is reasonable to ask questions such as :

Q.1 What correlated works of Molenaar and Jacobsen cover at least two topics from intelligence, language and mind?

Q.2 How is the concept of artificial intelligence presented in the context of language or mind, in current publications, without reference to articles about translation?

Existing works, even with a ranking function presented, cannot easily specify in Q.1 that "Molenaar" or "Jacobsen" is more important than "intelligence" or "language" or "mind", and only two out of the latter three is sufficient for a piece of collaborative talk to be returned; also there is no easy way to express the negative constraints in Q.2 concerning keyword "translation".

Moreover, frequently users would desire search results to contain data instances that are highly related to the query and at the same time bear *just the right amount of information* that satisfies their search, which means:

RQ.1 the result should not include non-relevant information;

RQ.2 the result should carry as rich information as possible;

RQ.3 the result should not include redundant information;

These requirements are not fully reflected by the ranking functions as proposed in [10–12, 14].

We take a different stand in this paper to address the open problems in both query specification and result definition as identified above. Rather than allowing users to give only one set of keywords as input, we propose the *ROU query*, which allows users to specify three different keyword sets: *Required*, *Optional*, and *Unwanted*. To our knowledge, even though the R and O concepts has been well exploited in keyword search and the concept of negation well studied in structured query language, we are the first in considering the explicit combination of them in keyword search on graph data. Our definition of the results of the the ROU queries not only focuses on fulfilling the keywords requirements of the three keyword sets, but also focuses on identifying results that provide richer yet non-reductant information.

To efficiently answer ROU queries, we propose a novel data structure, called query induced partite graph(QuIP), to capture candidate data entries that may contribute to query results, effectively transferring the problem of generating sub-graphs that fulfills the keyword constraints in the large data graph into the problem of enumerating cliques in QuIP. Then, inspired by the Bron-Kerbosch algorithm [2] and its variants [6, 16], we propose two algorithms which take advantage of the information captured in QuIP to efficiently generate results that maximize the keyword coverage and minimize the footprint.

Our contributions can be summarized as follows:

1. We propose the ROU query, a new keyword search problem on graph data, which integrates AND, OR and NOT semantics.
2. We propose the maximal-coverage and minimal-footprint semantics for ROU queries, which are well suited to be applied to other keyword search queries on graph data.
3. We propose a novel data structure, called query induced partite graph, and two algorithms utilizing query induced partite graphfor efficient ROU query evaluation.

2. PROBLEM DEFINITION

The data graphs we study in this paper are node-labeled undirected graphs $G = (V_G, E_G, \lambda_G)$, where V_G is the set of nodes, $E_G \subseteq V_G \times V_G$ is the set of edges, and $\lambda_G : V_G \rightarrow 2^{\mathcal{L}}$ is a labeling function that maps each node in V_G to a set of keyword terms in $\mathcal{L}$. We say a node v contains a keyword k if $k \in \lambda_G(v)$. We overload the mapping function $\lambda_G()$ such that when it is applied to a set of nodes $V \subseteq V_G$, it returns the union of the keyword sets of the nodes in V, i.e. $\lambda_G(V) = \bigcup_{v \in V} \lambda_G(v)$.

Our keyword search query allows users to specify both positive and negative keyword constraints, as well as size constraints. The results of our search queries are node sets satisfying such constraints and the requirements RQ. 1-3. We define the relationship between a set of nodes and keyword-based constraints, both positive and negative, as follows.

2.1 Size and Negative Keyword Constraints

Given a set of nodes V, when we consider it in the context of a data graph G, we measure the size of V by the distance among these nodes in G.

Definition 2.1. **[Distance & Diameter]** *Given a set of nodes $V \subseteq V_G$, for any two nodes $u, v \in V$, the distance between u and v, $dis(u, v)$, is the length of the shortest path between u and v in G. The diameter of V is*

$$d(V) = \max_{u,v \in V} dis(u, v)$$

Definition 2.2. **[Bypass Distance & Diameter]** *Given a set of nodes $V \subseteq V_G$ and a keyword set K, for any two nodes $u, v \in V$, the K-bypass distance of u and v, $\overline{dis}^K(u, v)$, is the length of the shortest path between u and v that does not pass through any node that contains a keyword in K. The K-bypass diameter of V is*

$$\overline{d}^K(V) = \max_{u,v \in V} \overline{dis}^K(u, v)$$

Please note that $\overline{dis}^K(u, v) = \infty$ if at least one keyword in K appears on each path between u and v [1]. Also observe that $\overline{dis}^{\{\}}(u, v) = dis(u, v)$.

Example 2.1. *Consider the sample data graph as shown in Figure 1. We have $dis(v_1, v_5) = 2$. However, given a keyword set $K = \{T\}$, $\overline{dis}^K(v_1, v_5) = 3$.*

[1]This includes the case in which $(\lambda_G(u) \cup \lambda_G(v)) \cap K \neq \emptyset$.

2.2 Positive Keyword Constraints

Consider a set of nodes that are close to each other and do not involve the unwanted keyword terms, we now move on to consider how to identify whether the set provides *just the right amount of information* desired by the user, given positive keyword constraints.

Definition 2.3. [**Cover**] *Given a set of nodes $V \subseteq V_G$ and a set of keyword terms K ($K \neq \emptyset$), we say V cover K (denoted $V \succ K$), if $K \subseteq \lambda_G(V)$ and for every $v \in V$, $\lambda_G(v) \cap K \neq \emptyset$.*

Definition 2.4. [**h-cover**] *Given a set of nodes $V \subseteq V_G$, a set of keyword terms K ($K \neq \emptyset$), and a threshold h, $0 \leq h \leq 1$, we say V h-cover K (denoted $V \succ^h K$), if $\frac{|K \cap \lambda_G(V)|}{|K|} \geq h$ and for every $v \in V$, $\lambda_G(v) \cap K \neq \emptyset$.*

Example 2.2. *Consider the sample data graph as shown in Figure 1 and a keyword set $K = \{A, I, L, M\}$, then, $\{v_3, v_4, v_5\} \succ K$, $\{v_3, v_5\} \not\succ K$, and $\{v_1, v_5\} \succ^{\frac{3}{4}} K$, $\{v_1, v_3\} \not\succ^{\frac{3}{4}} K$.*

The "*for every $v \in V$, $\lambda_G(v) \cap K \neq \emptyset$*" clause in both Def. 2.3 and Def. 2.4 ensures the satisfaction of RQ. 1.

RQ. 3 is fulfilled automatically when $|\lambda_G(v)| = 1$ holds for all nodes in G. However, when the label of a node contains multiple keyword terms, the situation becomes much more delicate. Existing works on keyword search in graph data [10,14], under such circumstances, chose to take a vague stand on how each node in the result represents the keywords, one or many, or as many as possible. Here, we will define the search results of the ROU problem in a more precise manner.

Symbol	**Description**
K_r,K_o,K_u	required, optional and unwanted keyword set
h	coverage threshold associated with K_o
dis_{MAX}	diameter constraint
$\overline{dis}^K(u, v)$	K-bypass distance between u and v
$V \succ K$	node set V cover keyword set K
$V \succ^h K$	node set V h-cover keyword set K
$V_1 <^K_{cv} V_2$	V_1's coverage of K is consumed by V_2's
$V_1 <^K_{fp} V_2$	V_1's footprint in covering K is smaller than V_2's
$\overbrace{q(G)}$	$q(G)$ under the *Maximal Coverage* semantics
$\underbrace{q(G)}$	$q(G)$ under the *Minimal Footprint* semantics
$\widehat{q(G)}$	$q(G)$ under the *Condensed* semantics
G_TqG	Query Induced Partite Graph (QuIP) of q on G
v^k	node in QuIP whose base is v in G and label is k
V^k_T	k-cluster in G_TqG

2.3 ROU Keyword Search Query

In the ROU problem we introduce, users can specify keyword constraints in three different categories: the set of keywords which they want to *all appear* in each result; the set of keywords which they want to *at least partially appear* in each result; and the set of keywords which *should not be associated with* the results. We call them *Required keyword set*, *Optional keyword set* and *Unwanted keyword set*, and use K_r, K_o and K_u to represent them, respectively. In addition, associated with the optional keyword set is a threshold h, which is a real number between 0 and 1. And a constraint dis_{MAX} needs to be specified to regulate the size of each result measured in terms of node set diameter.

The syntax and semantics of the ROU problem is defined as follows:

Definition 2.5. [**ROU Keyword Search Query**] *Given a data graph $G = (V_G, E_G, \lambda_G)$, an ROU query is specified in the form of $q = (K_r, K_o, K_u, h, dis_{MAX})$, in which* [2]

- *either K_r, K_o or K_u can be $\emptyset$;*
- *$K_r \cup K_o \neq \emptyset$;*
- *$0 \leq h \leq 1$ when $K_o \neq \emptyset$;*
- *$dis_{MAX} \geq 0$.*

The result of evaluating q on G is a set of node sets, i.e. $q(G) \in 2^{V_G}$. Each $V \in q(G)$ must satisfy:

1. *if $K_r \neq \emptyset$, then there exists $V_r \subseteq V$, such that $V_r \succ K_r$; otherwise, $V_r = \emptyset$;*
2. *if $K_o \neq \emptyset$, then there exists $V_o \subseteq V$, such that $V_o \succ^h K_o$; otherwise, $V_o = \emptyset$;*
3. *$V = V_r \cup V_o$;*
4. *$\overline{d}^{K_u}(V) \leq dis_{MAX}$.*

Example 2.3. *Consider Q1-2 presented in the introduction, they can be specified as ROU Keyword Search queries:*

Q.1: $q_1 = (\{Mo, Ja\}, \{I, L, M\}, \{\}, \frac{2}{3}, 3)$

Q.2: $q_2 = (\{A, I\}, \{L, M\}, \{T\}, \frac{1}{2}, 2)$

Applying q_2 to the data graph shown in Figure 1, we have

$$q_2(G) = \{\{v_1, v_2, v_4\}, \{v_1, v_2\}, \{v_2, v_4\}, \{v_3, v_5\}, \{v_2\}\}.$$

We would like to bring reader's attention to two types of subset relationships among node sets in $q_2(G)$:

Case 1. $\{v_1, v_2\} \subset \{v_1, v_2, v_4\}$, $\{v_1, v_2\} \succ \{A, I, L\}$, and $\{v_1, v_2, v_4\} \succ \{A, I, L, M\}$.

Case 2. $\{v_2, v_4\} \subset \{v_1, v_2, v_4\}$, $\{v_1, v_2, v_4\} \succ \{A, I, L, M\}$, and $\{v_2, v_4\} \succ \{A, I, L, M\}$.

As can be seen from the example above, some resultant node sets are consumed by others, hence: (1) a node set is by itself not rich enough, as $\{v_1, v_2\}$ in Case 1, violating RQ. 2; or (2) a node set contains redundant information, as $\{v_1, v_2, v_4\}$ in Case 2, violating RQ. 3.

Intuitively, when a set of keywords are partially covered (h-cover), the result that covers more keywords is considered to carry richer information comparing to the one that covers less, hence should be favored over the latter.

Definition 2.6. [**Coverage Comparison**] *Given a set of keywords K, consider two node sets $V, V' \subseteq V_G$. We say that V's coverage of K is consumed by V''s coverage of K, denoted $V <^K_{cv} V'$, if $V \subset V'$ and $\lambda_G(V) \cap K \subset \lambda_G(V') \cap K$.*

With the help of Def. 2.6, we can define the *maximal coverage* semantics as follows.

Definition 2.7. [**Maximal Coverage Semantics**] *Given a data graph $G = (V_G, E_G, \lambda_G)$, an ROU query $q = (K_r, K_o, K_u, h, dis_{MAX})$, the maximal coverage of the query result $q(G)$ is a subset of $q(G)$, defined as*

$$\overbrace{q(G)} = q(G) - \{V | \exists V' \in q(G)(V <^{K_r \cup K_o}_{cv} V')\}.$$

[2]We omit the condition $K_r \cap K_o = \emptyset$ as any query in which $K_r \cap K_o \neq \emptyset$ can be written into a query $q' = (K_r, K'_o, K_u, h', dis_{MAX})$, where $K'_o = K_o - K_r$, and h' is an adjusted threshold. We omit condition $K_r \cap K_u = \emptyset$, as any query that does not satisfy this condition has empty result. Another condition we omit is $K_o \cap K_u = \emptyset$, as a query that fails this condition can be rewritten into $q' = (K_r, K_o - K_u, K_u, h', dis_{MAX})$ with an adjusted threshold.

The maximal coverage semantics introduced above does not guarantee that each resultant node set is the minimal needed to cover the positive keywords ($K_r \cup K_o$) required in query q. As a remedy, we introduce the *minimal footprint* semantics, which ensures smallest resultant node set by favoring a smaller node set over a bigger one, when the two sets cover exactly the same sets of positive keyword terms. Formally,

Definition 2.8. **[Footprint Comparison]** *Given a set of keywords K, consider two node sets $V, V' \subseteq V_G$ which cover the same subset of K, i.e. $\lambda_G(V') \cap K = \lambda_G(V) \cap K$. If $V \subset V'$, we say that V has a smaller footprint than V' in covering K, denoted $V <^K_{fp} V'$.*

With the help of Def. 2.8, we can define the *minimal footprint* semantics as follows.

Definition 2.9. **[Minimal Footprint Semantics]** *Given a data graph $G = (V_G, E_G, \lambda_G)$, an ROU query $q = (K_r, K_o, K_u, h, dis_{MAX})$, the minimal footprint of the query result $q(G)$ is a subset of $q(G)$, defined as*

$$\underbrace{q(G)} = q(G) - \{V | \exists V' \in q(G)(V' <^{K_r \cup K_o}_{fp} V)\}.$$

Taking the concepts defined above all into consideration, we define the *condensed* semantics of ROU query, which satisfy RQ. 1, RQ. 2 and RQ. 3, as follows:

Definition 2.10. **[Condensed Semantics]** *Given a data graph $G = (V_G, E_G, \lambda_G)$, an ROU query $q = (K_r, K_o, K_u, h, dis_{MAX})$, the condensed semantics computes*

$$\widehat{q(G)} = \overbrace{q(G)} \cap \underbrace{q(G)}.$$

Example 2.4. *Consider again our example data graph and the search query q_2, we have*

$$\begin{aligned}\overbrace{q_2(G)} &= \{\{v_1, v_2, v_4\}, \{v_2, v_4\}, \{v_3, v_5\}\}\\ \underbrace{q_2(G)} &= \{\{v_2, v_4\}, \{v_3, v_5\}, \{v_2\}\}\\ \widehat{q_2(G)} &= \{\{v_2, v_4\}, \{v_3, v_5\}\}\end{aligned}$$

Note that a node set does not have to cover all the keywords in $K_r \bigcup K_o$ to qualify for $\overbrace{q(G)}$. $\{v_3, v_5\}$ is one such example.

Problem Definition: Given a data graph G and an ROU query q, find $\widehat{q(G)}$ efficiently.

3. QUERY INDUCED PARTITE GRAPH

From the definition of our ROU problem, a naïve approach for answering such queries, is to (1) identify keyword nodes, i.e. nodes that contain keyword terms; (2) identify node pairs among the keyword nodes that are within dis_{MAX} from each other; (3) construct a connectivity graph that contains nodes and edges that are resultant of step (2); (4) enumerate cliques on the connectivity graph. However, this approach is not applicable ro ROU queries.

The connectivity graph of query $q_2 = (\{A, I\}, \{L, M\}, \{T\}, \frac{1}{2}, 2)$, when applied on to our sample graph G, is shown in Fig. 2.

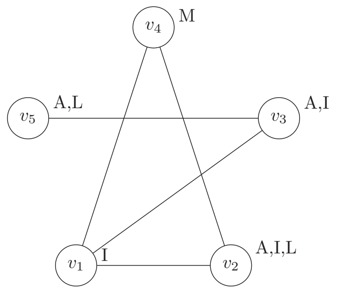


Figure 2: Connectivity Graph of $q_2(G)$

As shown in Example. 2.4, $\widehat{q_2(G)} = \{\{v_2, v_4\}, \{v_3, v_5\}\}$. While $\{v_3, v_5\}$ corresponds to a maximal clique in the connectivity graph, $\{v_2, v_4\}$ does not. That poses substantial difficulty to form a systematic enumeration algorithm. The connectivity graph as shown in Fig. 2 is not capable of capturing both the required and optional keyword requirements and our query semantics at the same time.

3.1 Introducing QuIP

We propose Query Induced Partite Graph (QuIP) as an intermediate data structure for efficient answering of ROU queries.

One key notion introduced in the QuIP is *shadowing.* Given a data graph $G = (V_G, E_G, \lambda_G)$ and a query $q = (K_r, K_o, K_u, h, dis_{MAX})$, for each node that covers a positive keyword in q, we create a copy of it for each positive keyword it covers, and label the copy with the keyword. These copies are the nodes in the QuIP. (We refer to the nodes and edges in QuIP as t-nodes and t-edges.) Two t-nodes are adjacent if their labels are different and (1) they are copies of the same original data node; or (2) the K_u-bypass distance between the two original data nodes they represent are within the size constraint (i.e. dis_{MAX}) in G. Formally,

Definition 3.1. **[Query Induced Partite Graph]** *Given a data graph $G = (V_G, E_G, \lambda_G)$ and an ROU query $q = (K_r, K_o, K_u, h, dis_{MAX})$, a* Query Induced Partite Graph *of q on G, denoted $G_T(q, G) = (V_T, E_T, \lambda_T)$ is constructed as follows:*

- *$V_T = \{v^k | v \in V_G \wedge k \in (K_r \cup K_o) \cap \lambda_G(v)\}$;*
- *$\lambda_T(v^k) = k$, for all $v^k \in V_T$;*
- *$(v^k, u^l) \in E_T$ if $k \neq l$ and (1) $v = u$; or (2) $v \neq u \wedge \overline{dis}^{K_u}(v, u) \leq dis_{MAX}$.*

Given a node v in data graph G, we call all t-nodes $v^k \in V_T$ the *shadows* of v. And for each such $v^k \in V_T$, we say that v is its *base.* We define the *base*() and *shadow*() functions to represent the mapping:

$$\begin{aligned} shadow(v) &= \{v^k | k \in (K_r \cup K_o) \cap \lambda_G(v)\}\\ base(v^k) &= v\\ base(V'_T) &= \bigcup_{v^k \in V'_T} base(v^k)\\ base(G'_T) &= base(V'_T) \text{ where } G'_T = (V'_T, E'_T, \lambda_T)\\ &\quad \text{is a subgraph of } G_T\\ base(S) &= \{base(G'_T) | G'_T \in S\}\\ &\quad \text{where S is a set of subgraphs of } G_T \end{aligned}$$

In a QuIP, given a keyword k, we call the set of all t-nodes labeled k the k-cluster, denoted $V^k_T = \{v^k | v^k \in V_T\}$.

Example 3.1. *Again consider query q_2 and sample graph G. $G_T(q_2, G)$ is shown in Figure 3. There are two shadow nodes of v_3: ${v_3}^A$ and ${v_3}^I$, i.e. shadow(v_3)= $\{{v_3}^A, {v_3}^I\}$. There are four k-clusters, shown in dotted circles.*

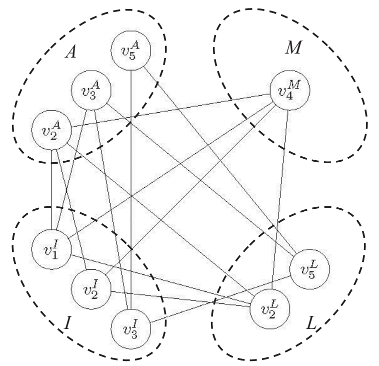


Figure 3: QuIP Graph of $q_2(G)$

The idea of grouping nodes based on the keywords they covered for enumeration purpose is used in some of the earlier works [10, 14], but their approaches did not explicitly create multiple copies for the same node because they only needed approximate solutions and did not pose precise semantics requirement. The concept of shadowing is essential for solving our problem as it adds both flexibility and constraints to enforce our query semantics.

The following properties of query induced partite graph can be established naturally from its definition.

Observation 3.1. *Given a QuIP $G_T(q, G) = (V_T, E_T, \lambda_T)$ of ROU query q on G, the following statements hold:*

- $V_T = \bigcup_{k \in K_r \cup K_o} V_T^k$;
- $V_T^k \cap V_T^{k'} = \emptyset$ *if* $k \neq k'$;
- *for any* $u^k, v^k \in V_T^k$, $(u^k, v^k) \notin E_T$.
- *given a t-node set* $V_T' \subseteq V_T$, *if the induced graph of* V_T' *is a clique, then,*
  - *for any two t-nodes* $v^k, u^l \in V_T'$, v^k, u^l *can not belong to the same keyword cluster, i.e.* $k \neq l$;
  - $|V_T'| \leq |K_r \cup K_o|$;

3.2 Answering ROU Queries Using QuIP

We now show how to take advantage of the information represented in a QuIP to efficiently answer an ROU query.

Example 3.2. *Let's again consider query q_2, whose QuIP $G_T(q_2, G)$ is shown in Figure 3.*

The answer to q_2, under the condensed semantics is $\widehat{q_2(G)} = \{\{v_2, v_4\}, \{v_3, v_5\}\}$. The two node sets are both bases of maximal cliques in $G_T qG$, with $base(\{{v_2}^A, {v_2}^I, {v_2}^L, {v_4}^M\}) = \{v_2, v_4\}$, and $base(\{{v_5}^A, {v_3}^I, {v_5}^L\}) = \{v_3, v_5\}$.

Note that different sets of t-nodes can be mapped to the same base. For example, consider $V_T' = \{{v_5}^A, {v_3}^I, {v_5}^L\}$ and $V_T'' = \{{v_3}^A, {v_3}^I, {v_5}^L\}$, while $V_T' \neq V_T''$, $base(V_T') = base(V_T'') = \{v_3, v_5\}$.

However, not all the maximal cliques in $G_T qG$ correspond to members of $\widehat{q(G)}$. For instance, $\{{v_3}^A, {v_1}^I\}$ is a maximal clique, but it does not contain enough keywords so its base $\{v_1, v_3\}$ does not even belong to $q(G)$. Another example is $\{{v_2}^A, {v_1}^I, {v_2}^L, {v_4}^M\}$. Its base is $\{v_1, v_2, v_4\}$ and satisfies maximal coverage semantics, however it does not satisfy the minimal footprint semantics as it covers no more keywords than $\{v_2, v_4\}$ does.

We summarize what we observed from Example 3.2 as follows:

Observation 3.2. *Given a QuIP $G_T(q, G) = (V_T, E_T, \lambda_T)$ of query q on G, the following statements hold:*

- *If node set $V \in \widehat{q(G)}$, then, there must exist a maximal clique c in $G_T qG$, such that $base(c) = V$.*
- *There may exist maximal clique c in $G_T qG$, but $base(c) \notin q(G)$.*
- *There may exist maximal cliques c, c' in $G_T qG$, such that $base(c) <_{fp}^{K} base(c')$, where $K = \lambda_T(base(c)) = \lambda_T(base(c'))$.*

Based on the above observations, finding all maximal cliques in QuIP may not suffice as an efficient manner for answering an ROU query. To address these issues, there is a special type of cliques in QuIP that interests us the most, the *h-cover clique.*

Definition 3.2. [**h-cover clique**] *Given a QuIP $G_T qG$, $G_T' = (V_T', E_T', \lambda_T)$ is a subgraph of G_T induced by t-node set V_T'. We say that G_T' is a h-cover clique of q if*

- *G_T' is a clique;*
- *There exists $V_{rT}' \subseteq V_T'$ and $V_{rT}' \succ K_r$, if $K_r \neq \emptyset$;*
- *There exists $V_{oT}' \subseteq V_T'$ and $V_{oT}' \succ^h K_o$, if $K_o \neq \emptyset$; and*
- *$V_T' = V_{rT}' \cup V_{oT}'$.*

Given a QuIP G_TqG, we use $hcClq(G_T qG)$ to represent the set of all h-cover cliques in G_TqG.

Theorem 3.1. *Given $G_T qG$, which is the QuIP of q on G, for any G_T' that is a sub-graph of $G_T qG$, the following holds:*

$$\widehat{q(G)} \subseteq base(hcClq(G_T qG)) \subseteq q(G).$$

Theorem 3.1 establish the QuIP as a suitable vehicle for answering ROU query. If we generate $hcClq(G_T qG)$, then, the base of $hcClq(G_T qG)$ is guaranteed to satisfy the positive and negative keyword constraints, but not necessarily satisfies the maximal coverage and minimal footprint semantics.

Let's consider two subsets of $hcClq(G_T qG)$:

$\overbrace{hcClq(G_T qG)}$ is a subset of $hcClq(G_T qG)$ that guarantees maximal coverage. Formally,

$$\begin{aligned}
\overbrace{hcClq(G_T qG)} \quad &= hcClq(G_T qG) \\
&-\{c | c \in hcClq(G_T qG) \\
&\quad \wedge \exists c' \in hcClq(G_T qG)(V_c \subset V_c' \\
&\quad \wedge \lambda_T(V_c) \cap K \subset \lambda_T(V_c') \cap K)\}
\end{aligned}$$

$hcC\widehat{lq(G_T}qG)$ is a subset of $\overbrace{hcClq(G_T qG)}$ that guarantees minimal footprint. Formally,

$$\begin{aligned}
hcC\widehat{lq(G_T}qG) \quad &= \overbrace{hcClq(G_T qG)} \\
&-\{mc | mc \in \overbrace{hcClq(G_T qG)} \\
&\quad \wedge \exists c \in hcClq(G_T qG)(base(c) \subset base(mc) \\
&\quad \wedge \lambda_T(c) \cap K = \lambda_T(mc) \cap K)\}
\end{aligned}$$

Corollary 3.1. *Given $G_T qG$, which is the QuIP of q on G, the following holds:*

$$base(\overbrace{hcClq(G_T qG)}) = \overbrace{q(G)}$$

$$base(hcC\widehat{lq(G_T}qG)) = \widehat{q(G)}.$$

EXAMPLE 3.3. *Again considering query q_2, and $G_T(q_2, G)$ as shown in Figure 3. Consider four sets of t-nodes: $V_{T_1} = \{{v_3}^A, {v_1}^I\}$, $V_{T_2} = \{{v_2}^A, {v_1}^I, {v_2}^L\}$, $V_{T_3} = \{{v_2}^A, {v_1}^I, {v_2}^L, {v_4}^M\}$, and $V_{T_4} = \{{v_2}^A, {v_2}^I, {v_2}^L, {v_4}^M\}$, each inducing a clique in $G_T(q_2, G)$, we call them $c_1 \ldots c_4$, respectively.*

$c_1 \notin hcClq(G_T(q_2, G))$ as V_{T_1} does not cover adequate keywords, and $base(V_T') = \{v_1, v_3\} \notin q(G)$.

$c_2 \in hcClq(G_T(q_2, G))$, however $c_2 \notin \overbrace{hcClq(G_T(q_2, G))}$ as c_2 is a sub-graph of c_3, which covers more keywords.

$c_3 \in \overbrace{hcClq(G_T(q_2, G))}$, however, $c_3 \notin hcClq(\widehat{G_T(q_2, G)})$, as c_3 and c_4 covers the same set of keywords, and $base(c_4) \subset base(c_3)$.

$c_4 \in hcClq(\widehat{G_T(q_2, G)})$, and $base(c_4) \in \widehat{q(G)}$.

Based on Theorem 3.1 and Corollary 3.1, we are ready to use the QuIP to answer an ROU query in two steps: (1) constructing QuIP based on an incoming query; and (2) identifying $hcClq(\widehat{G_T}qG)$ whose base is member of $\widehat{q(G)}$.

Both steps are very challenging, especially when the data graph is massive, or many data nodes carry more than one keyword terms, or many data nodes share the same keyword term, or the query is complicated, e.g. one or more keyword sets among K_r, K_o, and K_u are of decent size. In this paper, we focus on solving the problem related to step (2). While step (1) is not the focus of this document, we will first identify the challenges in QuIP construction, before presenting algorithms for computing query results using the QuIP in Section 4.

3.3 Challenges in QuIP Construction

The construction of QuIP involves identifying all the nodes V_T and edges E_T in G_TqG as defined in Def. 3.1.

An inverted index, which stores for each keyword term $k \in \mathcal{L}$ the set of IDs of the data nodes whose label contains k, can be used to assist the generation of V_T. Given a query q, a few index lookup will supply the sets of nodes that cover keyword terms in K_r and K_o, from which shadows can be constructed to form V_T.

Establishing the edges in E_T is much more challenging, due to the demand on $\overline{dis}^{K_u}(v, u) \leq dis_{MAX}$, in which, according to the definition of $\overline{dis}^{K_u}(v, u)$, *all* paths between u and v whose length is $\leq dis_{MAX}$ have to be checked to see if there exists at least one path that can bypass the keyword terms in K_u. It is impractical to check during runtime whether every path between a pair of nodes contains a certain keyword in K_u, especially when dis_{MAX} increases, not to mention that there may be thousands to millions pairs of nodes for each such conditions need to be checked.

Neighborhood index is often used to answer immediately whether two nodes are connected within a distance threshold [10]. It is possible to extend the neighborhood index to include labels. Such extension may help deduce whether two nodes are connected, bypassing a K_u of cardinality 1, however such extension is not sufficient for deducing whether they are connected, bypassing larger K_u. Designing summary data structures and algorithms for constructing QuIP efficiently is left as future work, and we consider cases where $|K_u| \leq 1$ in our experiments.

4. $\widehat{q(G)}$ GENERATION

4.1 Base Algorithm

Enumerating maximal cliques in a graph is a well studied problem in discrete mathematics [2, 16]. We adapt the Bron-Kerbosch algorithm [2] to find maximal cliques in QuIP. The details of the recursive algorithm is shown in Algo. 2. The process of generating the full set of $\widehat{q(G)}$ starts by triggering the `cdsCliqueEnum()` function, with P initialized to be V_T, and R and X empty set, is shown in Algo. 1.

Algorithm 1: $\widehat{q(G)}$ Generation

```
  Data: $G_T(q, G)$
  Result: $\widehat{q(G)}$
1 $R \leftarrow \emptyset$ ; $P \leftarrow V_T$ ; $X \leftarrow \emptyset$ ;
2 cdsCliqueEnum ($R$, $P$, $X$) ;
```

`cdsCliqueEnum()`, whose details are presented in Algo. 2, is amended with two pruning functions:

• `satisfyQ()` handles positive keyword constraints. Given a set of t-nodes $V \subseteq V_T$, `satisfyQ(V)=TRUE` if for some $V_r \subseteq V$, $V_r \succ K_r$ (if $K_r \neq \emptyset$) and for some $V_o \subseteq V$, $V_o \succ^h K_o$ (if $K_o \neq \emptyset$). It guarantees that all the maximal cliques enumerated by the algorithms are h-cover cliques.

• `notMnfp()` checks if there exists a node $v \in base(R)$ such that $\lambda_G(base(R)) \cap (K_r \cup K_o) = \lambda_G((base(R) - \{v\})) \cap (K_r \cup K_o)$. If so, the base of R cannot possibly be a subset of any results that satisfy the minimal footprint requirement as we can always remove v for a more "concise" result.

In Algo. 2, P is a set holding potential t-nodes that can be added to the current partial clique R. In the recursion process, t-nodes in P will always be neighbors to all the t-nodes in R. Thus whenever P is not empty, we can always pick a t-node from it to add to R to make a bigger clique. A t-node v is added to X only after all the maximal cliques containing $R \cup \{v\}$ are explored (line 12). Thus when P is empty and X is not, it implies that a clique that contains t-nodes in $R \cup X$ has already been considered before. So a clique that contains only t-nodes in R is not maximal, and our algorithm needs to backtrack. When both P and X are empty, a maximal clique in $G_T(q, G)$ has been formed; and it belongs to $hcClq(\widehat{G_T})$ because of `notMnfp()`. So we can report its base, which belongs to $\widehat{q(G)}$, as a query result. An execution example of the algorithm can be found in Table 1.

4.2 Optimization

Though `cdsCliqueEnum` generates exactly $\widehat{q(G)}$, it may consider cliques that consist of different t-nodes but share the same base, generating duplicated results. For example, applying the algorithm to $G_T(q_2, G)$, it will generate result $\{v_3, v_5\}$ twice, one from clique $\{{v_5}^A, {v_3}^I, {v_5}^L\}$, another from $\{{v_3}^A, {v_3}^I, {v_5}^L\}$.

To address this issue, we introduce `cdsCliqueEnumGreedy`, presented in Algo. 3, which can generate $\widehat{q(G)}$ without considering duplicated cliques. The basic idea is that when a t-node v is picked from P and put into R, we also put all shadows of the base of v that are in P, i.e. $P \cap shadow(base(v))$, into R (lines 9-11), and adjust P and X accordingly. When the search is done on the current level, all these t-nodes will be moved to X to prevent cliques with same base from being considered in the future (lines 15-17). Thus if we apply `cdsCliqueEnumGreedy`, assuming at the beginning in

Level	R	P	X	R'	P'	X'	Action
0	$\emptyset$	V_T	$\emptyset$	$\{v_3{}^I\}$	$\{v_3{}^A, v_5{}^A, v_5{}^L\}$	$\emptyset$	recursive call
1	$\{v_3{}^I\}$	$\{v_3{}^A, v_5{}^A, v_5{}^L\}$	$\emptyset$	$\{v_3{}^I, v_5{}^A\}$	$\{v_5{}^L\}$	$\emptyset$	recursive call
2	$\{v_3{}^I, v_5{}^A\}$	$\{v_5{}^L\}$	$\emptyset$	$\{v_3{}^I, v_5{}^A, v_5{}^L\}$	$\emptyset$	$\emptyset$	recursive call
3	$\{v_3{}^I, v_5{}^A, v_5{}^L\}$	$\emptyset$	$\emptyset$				report result, backtrack
2	$\{v_3{}^I, v_5{}^A\}$	$\emptyset$	$\{v_5{}^L\}$				backtrack
1	$\{v_3{}^I\}$	$\{v_3{}^A, v_5{}^L\}$	$\{v_5{}^A\}$	$\{v_3{}^I, v_3{}^A\}$	$\{v_5{}^L\}$	$\emptyset$	recursive call
2	$\{v_3{}^I, v_3{}^A\}$	$\{v_5{}^L\}$	$\emptyset$	...	...	...	...

Table 1: Evaluating q_2 Using Algo 2

Table 1 that we pick $v_3{}^A$ from P at level 0, then both $v_3{}^A$ and $v_3{}^I$ will be added to R. When the execution terminates, only clique induced by $\{v_3{}^A, v_3{}^I, v_5{}^L\}$ will be generated, and $\{v_5{}^A, v_3{}^I, v_5{}^L\}$ will be skipped.

Algorithm 2: cdsCliqueEnum(R,P,X)

```
   Data: $R, P, X$
   Result: bases of cliques that contain all t-nodes in R,
           some t-nodes in P and no t-nodes from X,
           satisfy condensed semantics
1  if notMnfp $(R)$ then
2      return
3  end
4  if $P$ is empty and $X$ is empty then
5      output $base$(R) as a result; return
6  end
7  for each t-node $v$ in $P$ do
8      $R' \leftarrow R \cup \{v\}$; $P' \leftarrow P \cap N(v)$; $X' \leftarrow X \cap N(v)$;
9      if satisfyQ $(R' \cup P')$ then
10         cdsCliqueEnum $(R', P', X')$
11     end
12     $P \leftarrow P - \{v\}$ ; $X \leftarrow X \cup \{v\}$ ;
13 end
```

Algorithm 3: cdsCliqueEnumGreedy(R,P,X)

```
   // Lines 1-6 same as Lines 1-6 in cdsCliqueEnum
7  for each t-node $v$ in $P$ do
8      $R' \leftarrow R$; $P' \leftarrow P$; $X' \leftarrow X$;
9      for each t-node $v'$ in $P \cap$ $shadow(base(v))$ do
10         $R' \leftarrow R' \cup \{v'\}$; $P' \leftarrow P' \cap N(v')$;
           $X' \leftarrow X' \cap N(v')$;
11     end
12     if satisfyQ $(R' \cup P')$ then
13         cdsCliqueEnum $(R', P', X')$
14     end
15     for each t-node $v'$ in $P \cap$ $shadow(base(v))$ do
16         $P \leftarrow P - \{v'\}$ ; $X \leftarrow X \cup \{v'\}$ ;
17     end
18 end
```

4.3 Discussion

Theoretically the complexity of enumerating maximal cliques in a graph with n nodes is $O(3^{\frac{n}{3}})$, as stated in [15]. However, the complexity of our algorithms is much lower. In the worst case, the complexity is $O((\max_{k \in K_r \bigcup K_o} |V_T^k|)^{|K_r \bigcup K_o|})$, where $|V_T^k| \lll |V_G|$ and $|K_r \bigcup K_o|$ is small. Moreover, due to the partite nature and sparsity of $G_T(q, G)$, it is often the case that in $G_T(q, G)$ each node is adjacent to only a few nodes. Under such circumstance, the recursion tree is shallow and subtrees are small below the first level of recursion. Hence, the average complexity is very close to $O(|V_T|)$.

5. EXPERIMENTAL EVALUATIONS

5.1 Experimental Setup

We conduct our experimental evaluation on the Proximity DBLP database, in which nodes represents authors and publications and edges represent *cites* and *author-of* relationship between corresponding node pairs. The Proximity DBLP database is based on data from the DBLP Computer Science Bibliography with additional preparation performed by the Knowledge Discovery Laboratory, University of Massachusetts Amherst. As a proof-of-concept experiments, the data contains 1.1M nodes and 1.8M edges, and the total number of node pairs (u, v) that satisfies $dis(u, v) \leq 3$ is 111M.

The keyword terms bear various single node frequency in the dataset. For instance, more than 1% of nodes contain the keywords "system", "parallel" or "learning", and less than 0.2% contains "robotics", "sql" or "compiler". Apriori knowledge of the field of computer science helps interpret term closeness, for instance, the terms "database" and "query" are relatively close, while the relationship between "cryptography" and "robotics" is much more distant. Based on the term closeness, we organize terms in groups, then, when keywords are selected to compose ROU queries, we are able to form queries that are representative enough to better interpret the impact of various of input parameters. We will present results of ROU queries whose keywords are picked from those in the table below.

Group	Keywords
G1	bayesian, inference, statistical, polynomial, propagation, belief, graphical
G2	artificial, intelligence, reasoning, vision, learning, robotics
G3	compiler, type, programming, semantics, grammar
G4	database, query, optimization, sql, datalog
G5	operating, system, parallel, distributed, memory

Our algorithms are written in Java and our experiments are run on a laptop running Windows 7 with Intel(R) Core(TM) i5-2450M CPU @ 2.50GHz and 8GB memory, in which a maximum of 4GB memory are dedicated to the Java virtual machine.

5.2 QuIP Size

We first examine the size of the QuIP, which is the prime vehicle in our proposed techniques for answering ROU queries. The QuIP sizes of some example queries (with $dis_{MAX}= 3$) are shown in the table below.

Query	$K_r \cup K_o$		K_u		Nodes	Edges
	$\lvert K_r \cup K_o \rvert$	groups	$\lvert K_u \rvert$	groups		
Q_1	6	G1	0		13743	61538
Q'_1	6	G1	1	G1	13498	58208
Q_2	5	G2	0		13576	35713
Q'_2	5	G2	1	G2	13464	35362
Q_3	5	G3	0		12037	132K
Q_4	5	G4	0		13764	2.5M
Q'_4	5	G4	1	G4	12853	1.8M
Q_5	5	G5	0		56976	1.2M
Q_6	5	G1 to G5	0		8262	79K

Observations:

- In general, QuIP is significantly smaller than the data graph, both in terms of node number and edge number, considering the fact that the data graph has over 1M nodes and over 111M node pairs whose distance is less than 3.
- The presence of negative keyword constraint, even when the size of the keyword set is 1, can reduce the size of $QuIP$ (and subsequently lead to decrease of enumeration time).

Naturally, graph algorithms are very sensitive to the size of the input data. In the following, we are presenting results on a variety of graphs that are of different size, i.e. graphs generated by Q_1 to Q_5.

5.3 Algorithm Performance Comparisons

We compare the performance of the two query answering algorithms we proposed, i.e. one using recursive function `cdsCliqueEnum`, another using `cdsCliqueEnumGreedy`, and with the baseline algorithm, which use the Bron-Kerbosch algorithm to generate all maximal cliques on the QuIP, then applying filtering to verify the keyword constraints and the condensed semantics. The performance comparison of a selected set of queries is shown in Figure 4. Please note that log-scale is used for y-axis. For Query 3-5, we cut off the computation at 100K results as the potential result size can be huge.

Observations:

- The `cdsCliqueEnumGreedy` algorithm generally outperforms the baseline algorithm, cutting execution time by half. In cases, such as Q_4, where few among large number of maximal cliques in the QuIP satisfy the keyword constraints and condensed semantics, the pruning is particularly effective and outperforms the baseline algorithm by two orders of magnitude (more than 100 times faster).
- The `cdsCliqueEnum` algorithm has comparable performance as the baseline. However, the pruning that is pushed into the clique generation process can significantly out-perform the baseline in worst-case scenario, such as Q_4.

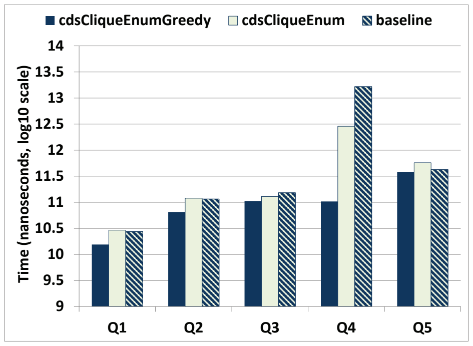


Figure 4: Algorithms Performance Comparisons

- Theoretically, `cdsCliqueEnum` algorithm may outperform `cdsCliqueEnumGreedy` when the overhead of the additional pruning employed by the later overshadow the benefit of such pruning, However, such cases are extremely rare according to our experimental study.

5.4 Impact of Positive Keyword Constraints

Positive keyword constraints impact the result size, hence the time spent to evaluate the query, from multiple angles. As shown earlier when we discussed the size of QuIP, the more positive keywords are in the query, the larger the QuIP; and with the same number of positive keywords, the closer the semantic relationship among the positive keywords, the larger the QuIP.

Here, we study the impact of the distribution of the keywords among K_r and K_o on result size and query performance. To measure this impact, we introduce the notion of *valid keyword combinations count*, which is a function of K_o and h, i.e. $Ct(K_o, h) = \sum_{n=\lceil |K_o| \times h \rceil \ldots |K_o|} C^n_{|K_o|}$. When two queries have the same set of positive keywords, hence the same QuIP, the larger the valid keyword combination count, the larger the result size, the more maximal cliques need to be exploit in answering the queries, hence the longer the run time.

We randomly pick sets of keyword terms, and for each set, vary the distribution of these keywords among K_r and K_o, and the h value, then, compare the result sizes and query evaluation time among the queries that shares the same $K_r \cup K_o$. The heuristics stated above is proved in all cases. Figure 5 shows the results on a set of 6 keywords picked from group 2.

Please note that when the valid keyword combinations count increases, the evaluation time increases at a much slower pace than the result size. This is due to the fact that the cliques exploited in the evaluation process share common sub-clique, and the recursive nature of our algorithms are able to absorb the extra computational cost.

6. CONCLUSION AND FUTURE WORK

In this paper we introduced ROU, a new type of keyword search queries on graph data, which allows user to specify both positive (including required and optional) and negative keyword constraints. We formally defined the semantics of such query in terms of maximal coverage and minimal footprint. We proposed a data structure, called query induced

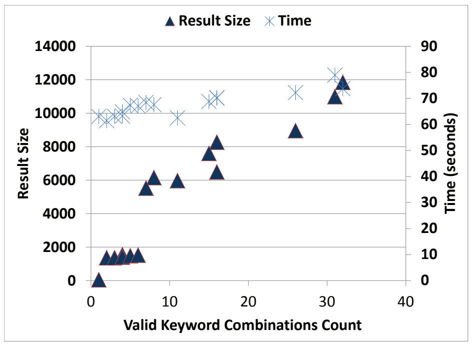


Figure 5: Impact of Positive Keywords

partite graph (QuIP) for representing candidate data entries and their relationships, and algorithms that take advantage of information collected in QuIP to efficiently answer ROU queries efficiently.

The problem of ROU is far from solved. Natural extension to the work presented in this paper include: (1) inventing new summary data structures and indexing techniques to efficiently construct QuIP especially for negative keyword conditions; (2) introducing ranking methods to identify results that are most interesting to users and develop approximation semantics and algorithms for identifying top-k results efficiently; (3) exploring parallel implementations of our algorithms to ensure that our solution scale well in terms of both data size and query complexity; and (4) extending query semantics for other types of interesting keyword queries.

7. REFERENCES

[1] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword Searching and Browsing in Databases using BANKS. In *ICDE*, pages 431–440, 2002.

[2] C. Bron and J. Kerbosch. Finding all cliques of an undirected graph (algorithm 457). *Commun. ACM*, 16(9):575–576, 1973.

[3] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSEarch: A Semantic Search Engine for XML. In *VLDB*, pages 45–56, 2003.

[4] B. B. Dalvi, M. Kshirsagar, and S. Sudarshan. Keyword search on external memory data graphs. *PVLDB*, 1(1):1189–1204, 2008.

[5] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin. Finding top-k min-cost connected trees in databases. In *ICDE*, pages 836–845, 2007.

[6] D. Eppstein and D. Strash. Listing all maximal cliques in large sparse real-world graphs. In *SEA*, pages 364–375, 2011.

[7] K. Golenberg, B. Kimelfeld, and Y. Sagiv. Keyword proximity search in complex data graphs. In *SIGMOD*, pages 927–940, 2008.

[8] H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS: ranked keyword searches on graphs. In *SIGMOD*, pages 305–316, 2007.

[9] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient ir-style keyword search over relational databases. In *VLDB*, pages 850–861, 2003.

[10] M. Kargar and A. An. Keyword search in graphs: Finding r-cliques. *PVLDB*, 4(10):681–692, 2011.

[11] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *SIGMOD*, pages 903–914, 2008.

[12] F. Liu, C. T. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *SIGMOD*, pages 563–574, 2006.

[13] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval.* Cambridge University Press, 2008.

[14] L. Qin, J. X. Yu, L. Chang, and Y. Tao. Querying communities in relational databases. In *ICDE*, pages 724–735, 2009.

[15] E. Tomita, A. Tanaka, and H. Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theor. Comput. Sci.*, 363(1):28–42, 2006.

[16] M. J. Zaki, M. Peters, I. Assent, and T. Seidl. Clicks: An effective algorithm for mining subspace clusters in categorical datasets. *Data Knowl. Eng.*, 60(1):51–70, 2007.