Journal of Informetrics 10 (2016) 1092–1107

Contents lists available at ScienceDirect

Journal of Informetrics

journal homepage: www.elsevier.com/locate/joi

# Searching bibliographic data using graphs: A visual graph query interface

Yongjun Zhu*, Erjia Yan

*College of Computing and Informatics, Drexel University, 3141 Chestnut Street, Philadelphia, PA 19104, United States*

ARTICLE INFO

*Article history:*
Received 12 May 2016
Received in revised form 19 August 2016
Accepted 24 September 2016
Available online 6 October 2016

*Keywords:*
Information retrieval
Bibliographic queries
Graph query interface
Information visualization
Graph databases

ABSTRACT

With the ever-increasing scientific literature, improving the efficiency of searching bibliographic data has become an important issue. With a lack of support of current bibliographic information retrieval systems in expressing complicated information needs, getting relevant bibliographic data is a demanding task. In this paper, we propose a visual graph query interface for bibliographic information retrieval. Through this interface, users can formulate bibliographic queries by interacting with a graph. Visual graph queries use a set of nodes with constraints and links among nodes to represent explicit and precise bibliographic information needs. The proposed visual graph query interface allows users to formulate several complex bibliographic queries (e.g., bibliographic coupling) that are not attainable in current major bibliographic information retrieval systems. In addition, the proposed interface requires less number of queries in completing everyday bibliographic search tasks.



## 1. Introduction

Graph data are prevalent in the real world as data from a variety of domains (e.g., physics, chemistry, biology, sociology, and computer science) can be represented by graph data models (Aggarwal & Wang, 2010). Graph data models can represent relational information and enable a number of applications by supporting efficient searching and mining (Cook & Holder, 2006). Because of this, a few studies have investigated ways of generating graphs from arbitrary data (e.g., Baeza-Yates, Brisaboa, & Larriba-Pey, 2010). Bibliographic data are graph data in nature because they can be represented in the form of interconnected papers, authors, terms, sources, and organizations. Recent bibliometric studies, including searching bibliographic data (Zhu, Yan, & Song, 2016), measuring scholarly impact (Yan & Ding, 2009), and mining bibliographic networks (Sun, Barber, Gupta, Aggarwal, & Han, 2011) have taken the advantage of the graphical representation of bibliographic data. Regardless of the physical representations (e.g., relational databases) of graph data, efficient searching of graph data is one of primary tasks for the information retrieval community (e.g., Kacholia et al., 2005; Jiang, Wang, Yu, & Zhou, 2007; Yuan, Wang, Chen, & Wang, 2013). User interface is an integral part of searching (Hearst, 2009), and a variety of user interfaces have been proposed to support searching graph data, including regular expression- (Giugno & Shasha, 2002), keyword- (Tran, Wang, Rudolph, & Cimiano, 2009), and natural language-based (Pradel, 2012) interfaces. A recent movement towards efficient graph data searching is to adopt graph queries (e.g., Zhang, Zhang, Tang, Rao, & Tang, 2010; Han, Finin, & Joshi, 2012). Graph queries are a way of searching graph data by taking a graph pattern with a few constraints over nodes and edges as input,

* Corresponding author.
*E-mail addresses:* zhu@drexel.edu (Y. Zhu), ey86@drexel.edu (E. Yan).

http://dx.doi.org/10.1016/j.joi.2016.09.005
1751-1577/

which is a natural fit to graph data (He and Singh, 2008). Graph queries are known to convey richer information than other forms of queries and thus improve search performance (e.g., Zhou, Wang, Xiong, Wang, & Yu, 2008). Traditionally, systems with a graph query interface relied on textually represented graph queries (i.e., graph query languages). For example, He and Singh (2008) proposed a graph algebra-based query language to explore graph data, and, likewise, Leser (2005) proposed a pathway query language for biological networks. Even though textually represented graph queries are an effective way of searching graph data, writing these queries requires substantial efforts and causes a hindrance to users (Jayaram, Khan, Li, Yan, & Elmasri, 2015). As an alternative, a few studies proposed searching graph data using visually represented graph queries (e.g. Ceri, Comai, Damiani, & Fraternali, 1999). These visual graph queries are seen as more user-friendly because users do not need to remember the syntax of textual graph queries (Ykhlef & Alqahtani, 2011).

Despite the improved performance of graph queries, searching bibliographic data still faces a critical challenge-the proliferation of data has made it increasingly burdensome to retrieve relevant literature. Major bibliographic search systems provide forms, keywords, and Boolean queries as the main interfaces for searching bibliographic data. A typical search scenario is that users need to go through multiple refining processes after sending the first query; even so, they usually end up in getting too many search results than can be absorbed. Therefore, it is imperative to enable queries to covey more explicit information and represent more complicated information needs to only return the most pertinent search results. Aforementioned search user interfaces have limitations in representing such precise information needs. For example, they cannot directly represent queries such as "*papers on information retrieval, which were cited by John's papers that had been presented in SIGIR*". This type of queries, on the other hand, can be easily represented by visual graph queries with a set of nodes with constraints and links. Visual graph query interfaces are thus seen as a reliable solution for users to express precise and explicit information needs and receive more relevant search results. With this motivation in mind, this study aims to propose a visual graph query interface for bibliographic information retrieval. Specifically, this study aims to address the following research questions:

- How to design and implement a visual graph query interface to search bibliographic data?
- What features does a visual graph query interface need to have in order to improve bibliographic data retrieval? And
- How to integrate a visual graph query interface with back-end databases to build a streamlined system?

The work is built on our previous work (Zhu et al., 2016), in which, we proposed a framework for graph-based bibliographic information retrieval. In the present work, we focus on visual graph query formulation and processing while using the same graph schema proposed in our previous work.

## 2. Literature review

Earlier studies on visual graph queries were carried out by taking a specific data structure—XML-in mind (e.g., Ceri et al., 1999; Erwig, 2003; Ni & Ling, 2003; Ykhlef & Alqahtani, 2009). These studies proposed visual graph queries for querying and restructuring XML data. As XML data are quite complex with multiple nested structures, visual graph queries are seen as an efficient solution. Because the main goal of these studies was to build efficient languages of visual graph queries by investigating the structural aspects of XML documents, they are intended to be used by other systems but not the end users.

Recent studies (Hogenboom et al., 2010Hogenboom, Milea, Frasincar, & Kaymak, 2010; Schweiger, Trajanoski, & Pabinger, 2014) proposed visual graph query interfaces for users to query graph data. However, these visual graph queries were designed only to search for data that are stored as Resource Description Framework (RDF) triples, which is a standard data format of Semantic Web. Because SPARQL is the de facto standard RDF query language, those visual graph queries were designed to be translated into SPARQL, which limits their applicability. The proposed visual graph query interface in this study, however, does not stick to a specific database, and is more flexible (e.g., relational databases, graph databases).

Besides the abovementioned differences, a clear distinction between our study and the earlier studies is that we focus on a specific domain—bibliographic information retrieval. By focusing on this domain, we are able to deliver more customized solutions that can be used in real-world cases. To the best of our knowledge, the only one that is the most relevant to our study is the study by Gómez-Villamor et al. (2008). They proposed a bibliographic exploration tool based on a graph query engine. The tool employed visual graphs, while the actual queries are formulated by clicking one of three predefined queries other than a graph. But the work is largely different from our study in that we propose visual graph queries for bibliographic information retrieval, in which users visually draw graph queries to search bibliographic data.

## 3. Bibliographic graph queries

Bibliographic data are by nature a directed graph with nodes and links. For example, a link named "*WRITES*" is a directed link, in which the source is an "*Author*" and the target is a "*Paper*". There are a variety of ways to model bibliographic data using graphs. The one shown in Fig. 1 shows a typical schema of bibliographic data with five bibliographic entities. In the schema, "*Source*" denotes a journal or a conference in which authors publish or present papers. "*Term*" denotes a keyword, a topic; or a concept that describes a paper. In the rest of this paper; we will use this typical schema to explain the visual graph query interface.

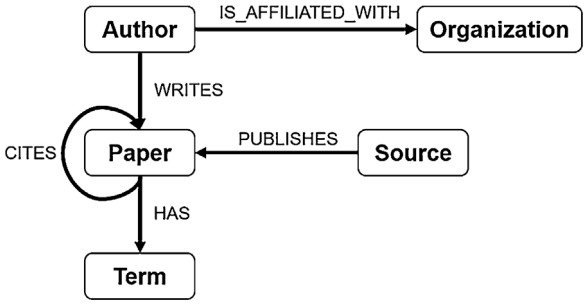



**Fig. 1.** A graph schema of bibliographic data.

Source: Zhu et al., (2016).

It is possible to model bibliographic data using different schemas. For instance, one can model it as a multigraph, in which there are multiple relations between two bibliographic entities. One possible scenario is that paper A cites paper B, and paper A is also co-cited with paper B. In this scenario, there are two types of relations between paper A and paper B: citation and co-citation. However, this scenario can be well represented with the current schema by introducing another paper—paper C, i.e., paper C cites both paper A and paper B. In this case, the co-citation relation between paper A and paper B is indirectly represented through paper C. When creating the schema, we explicitly modeled the schema as simple as possible. There are two reasons for this: first, users need to formulate queries after learning the schema, and a simple schema would expedite the learning process; second, a simple schema does not hurt users' imaginative power, and allows more variants of representations.

Bibliographic graph queries can be formulated on the basis of this schema with the following additional information:

### *3.1. Node type*

Every node in a graph query needs to be specified with a type (e.g., *Paper*). Node type is essential for creating links among nodes because links are created by considering the relations of the types of two nodes. For instance, a node with the type of "*Organization*" is not linked with a node with the type of "*Term*", while it can be linked with a node with the type of "*Author*".

### *3.2. Answer node*

An answer node is the node that answers a visual graph query. For instance, in a visual graph query "*papers on information retrieval that were written by Salton*", the answer node is "*paper*", because the query is asking for returning papers as the final search result. A visual graph query should have at least one answer node. This means that we can retrieve bibliographic entities with more than one type by formulating proper visual graph queries.

### *3.3. Constraint node*

A constraint node denotes a node that constrains a visual graph query. In the above example query, a node with the name "*Salton*" is a constraint node that restricts the query to retrieve only papers written by "*Salton*". A visual graph query includes one or more constraint nodes.

### *3.4. Node name*

Unlike answer nodes, constraint nodes should have names in addition to types. Node names are only assigned to constraint nodes because regular nodes do not need names to constrain the query.

Fig. 2 shows four example bibliographic graph queries with varying lengths (i.e., the number of nodes and links), in which the answer nodes are in black and constraints nodes are in red. Regular nodes that connect answer nodes and constraint nodes are in blue. Directions of the links are based on the schema shown in Fig. 1.

As shown in Fig. 2, bibliographic graph queries are visually represented, and the proposed visual graph query interface is intended to process these visual bibliographic queries drawn by users.

## 4. Methods

In this section, we discuss the methods of processing visual graph queries. First, we present the designed system architecture and show the process flow and interconnected components. Then, we introduce and discuss with example queries the components of visual graph queries.

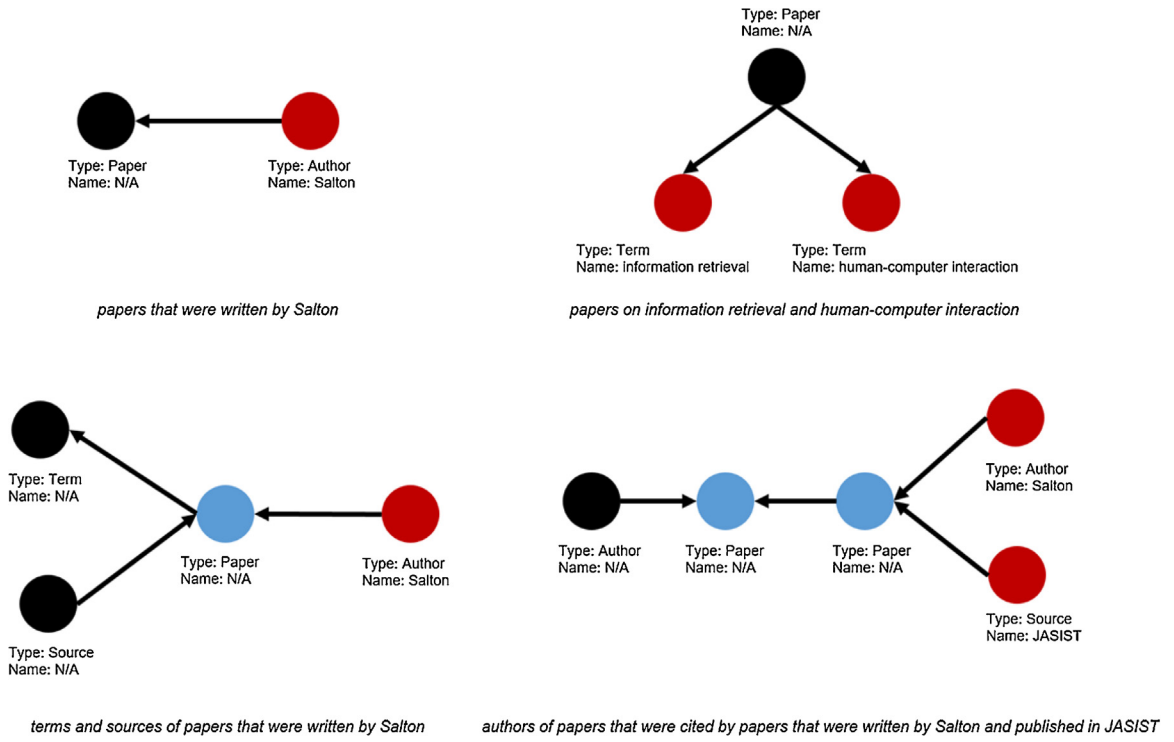



**Fig. 2.** Four example visual graph queries.

### 4.1. System architecture

The processing of visual graph queries begins with a verification stage because users can create erroneous queries. The verification stage includes both syntax checking and semantics checking. Syntax checking examines whether a visual graph query includes all necessary constructs (i.e., node type, answer node, constraint node, and node name). A syntactically correct graph query is not necessarily a meaningful query, and requires semantics checking. Semantics checking examines whether a visual graph query is answerable by checking the structure of the visual graph query. In this stage, incorrect queries are corrected and ambiguous queries are identified. Possible interpretations of ambiguous queries are sent back to users for their confirmation. Last, a verified visual graph query is translated into a database query to search in a database. Fig. 3 shows the architecture of a visual graph query-based bibliographic information retrieval system.

In the following sections, we discuss in detail the methods involved in the verification of visual graph queries, the generation of candidate graph queries, and the interpretation of graph queries.

### 4.2. The verification of visual graph queries

To satisfy syntax checking requirements, 1) a visual graph query should be a single graph with every single node having at least one relation with other nodes, 2) every node should have a node type, 3) the visual graph query should have at least one answer node, 4) the visual graph query should have at least one constraint node, and 5) all constraint nodes should have names.

In regards to semantic checking, every semantically incorrect link can be divided into three categories (i.e., shortest path equals to zero, shortest path equals to one, and shortest path greater than one) based on the length of the shortest path between the source and the target of the link. Shortest paths are obtained from the data schema of bibliographic data (Fig. 1). We apply the following algorithm (Fig. 4) to a visual graph query for semantics checking. We perform query correction and query disambiguation for syntactically incorrect queries. We check every link of a visual graph query and perform query correction and disambiguation on the basis of links.

As shown in Fig. 4, every link of a visual graph query is checked for the length of shortest path between the source and the target. Any link that does not conform to the schema is then updated. Query correction is performed when the direction of a link is incorrect (e.g., from a paper node to an author node) or two nodes are connected when they are not directly related on the schema (e.g., a link from a paper node to an organization node). While query correction is achieved for two of three categories (i.e., shortest path equals to one and shortest path greater than one); query disambiguation is accomplished for all three categories to generate all possible candidate graph queries.

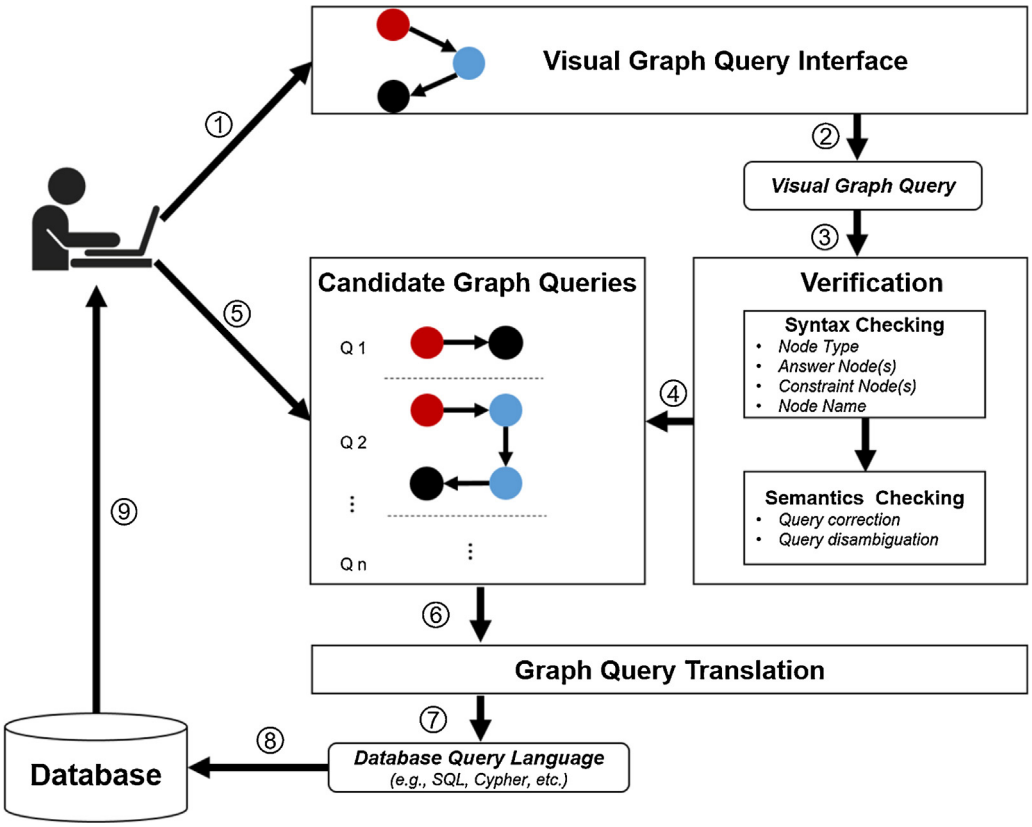



**Fig. 3.** The architecture of a visual graph query-based bibliographic information retrieval system.

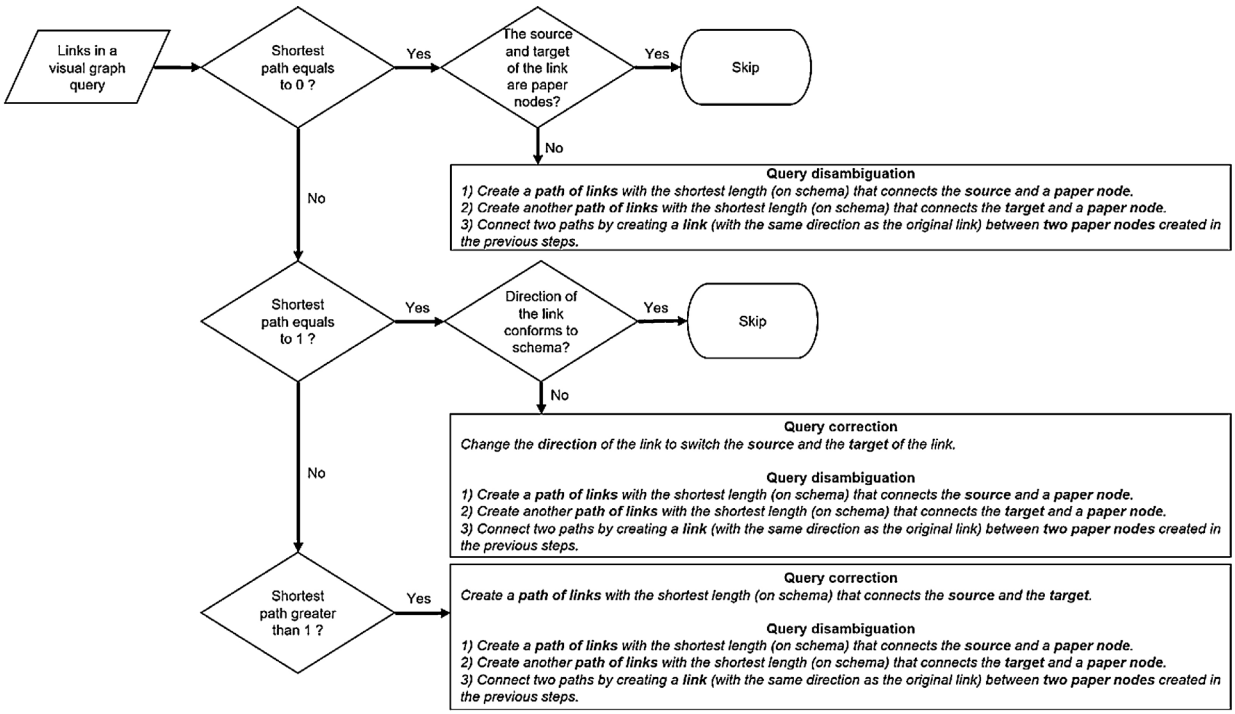



**Fig. 4.** An algorithm for query semantics checking.

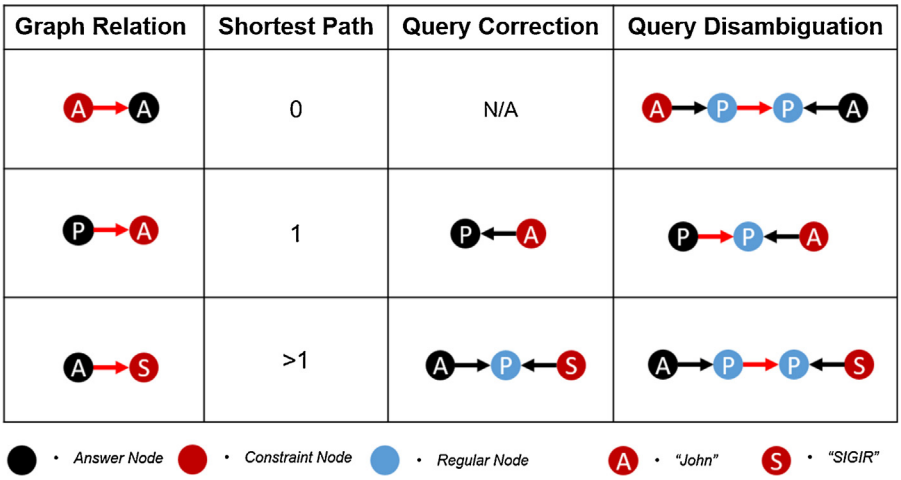



**Fig. 5.** Examples of query correction and disambiguation.

Fig. 5 shows three examples of semantically incorrect links as well as how they are corrected and disambiguated. Letters inside the nodes denotes node types (i.e., A for author, P for paper, T for term, S for source, and O for organization). Names of constraint nodes (red nodes) are shown under the table.

If the source and the target of a link are the same, we treat the shortest path between the source and the target as zero. In the first example, both the source and the target is *Author*. This is semantically incorrect because there is no link between the two author nodes based on the schema. Even though the first example is incorrect in semantics, it could be formulated by users with a specific meaning, i.e., *authors cited by John*, which should be *authors who wrote papers that were cited by papers written by John*. Thus, for a link with shortest path equals to zero, we disambiguate the link by treating the link as a citation relation and adding two additional *Paper* nodes as shown in the last column of the first example. Two *Paper* nodes are connected with the source and target respectively by finding their shortest paths from the source node and to the target node. Finally, two paper nodes are connected with a link that has the same direction as the original link because the direction of the original link is considered as the direction of the citation relation. For the second example, the direction of the link is opposite to that of the schema: in the schema, the link between a *Paper* and an *Author* is from an *Author* to a *Paper* with the link label of *WRITES*. Thus, we correct the link by changing the direction of the link. It is also possible that the user may mean *papers cited John* (i.e., *papers cited papers that were written by John*). Thus, query disambiguation is performed by creating a new graph query with the above meaning. Similarly, the third example can either mean *authors that presented papers in SIGIR* or *authors who wrote papers that cite papers that were presented in SIGIR*. Query correction and disambiguation are performed, and two new paths are created. In this way, query correction and disambiguation helps users formulate correct visual graph queries.

### 4.3. The generation of candidate graph queries

Candidate graph queries are the enriched version of visual graph queries, in which links are added with labels. Candidate graph queries are generated as the results of the verification of visual graph queries. If a user formulates a syntactically and semantically correct visual graph query, there would be only one candidate graph query. Otherwise, there would be more than one candidate graph query. As shown in Fig. 5, a semantically incorrect link resulted in one or two paths/links through the process of semantics checking. Because a visual graph query comprises one or more links, if a visual graph query includes two semantically incorrect links, there would be up to four possible interpretations. If a query includes $n$ incorrect links, the maximum number of candidate graph queries would be $2^n$. Given a visual graph query, we generate all possible candidate graph queries and return them back to the user who formulated the visual graph query. The user then selects one that best represents his/her information needs to proceed. Fig. 6 shows a visual graph query with two semantically incorrect links (dotted links) and candidate graph queries generated from the visual graph query.

As shown in Fig. 6, each semantically incorrect link resulted in two possible interpretations by ways of query correction and disambiguation. We combined all possible interpretations and generated four candidate graph queries. The user can select one to proceed. The next step is to translate the selected graph query into a database query.

### 4.4. The interpretation of graph queries

A graph query needs to be translated into a database query to retrieve data in a database as shown in Fig. 3. We choose two types of popular databases (i.e., a relational database and a graph database) for bibliographic data management, and discuss how a graph query is translated into SQL (for relational databases) and Cypher (for graph databases). Relational

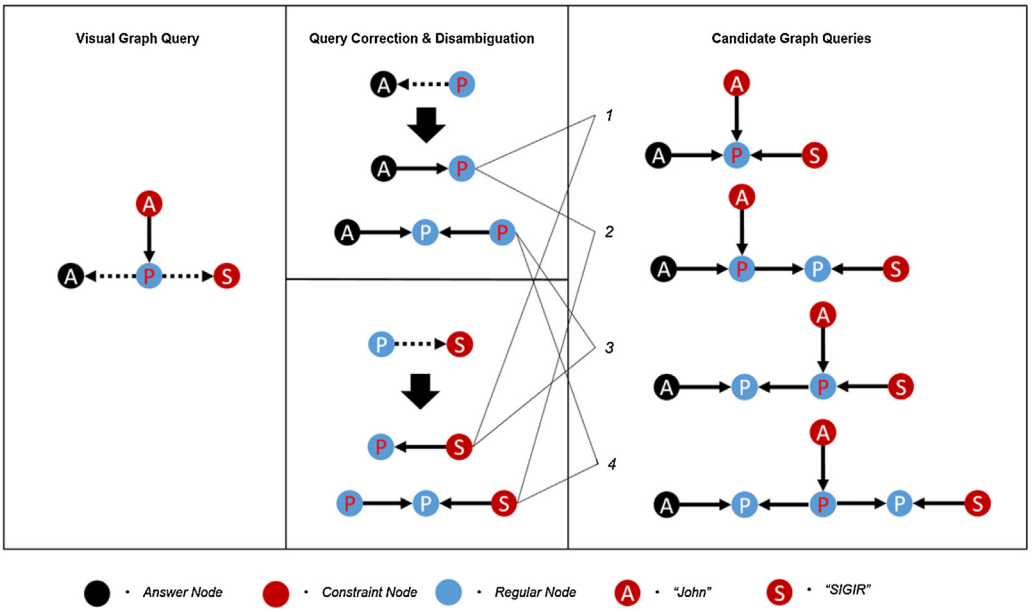



**Fig. 6.** An example of generating candidate graph queries.

databases are the most popular databases for information retrieval systems (Vicknair et al., 2010) while graph databases support graph data model for managing bibliographic data and have better performance in retrieving those data (Zhu et al., 2016).

*4.4.1. The translation of graph queries into SQL commands*

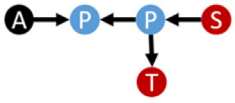
We discuss how graph queries are translated into SQL commands to query a database that is implemented based on the ER model (Fig. A1 in Appendix A). We show how three constructs (i.e., *SELECT*, *FROM*, and *WHERE*) of SQL are created from graph queries. Fig. 7 shows four example graph queries and their translations into SQL commands.

A *SELECT* clause defines the data that we want to retrieve. It is straightforward as shown in Fig. 7. The name(s) of the answer node(s) are the data that we want to retrieve, thus forming a *SELECT* clause. We specify tables from which to retrieve the data in a *FROM* clause. A *FROM* clause includes more tables than nodes in a graph query because of foreign keys. Non-intermediate tables (i.e., tables for bibliographic entities) included in a *FROM* clause are tables for answers node(s) and constraint node(s). Thus, the number of non-intermediate tables in a *FROM* clause is the same as the number of answer node(s) and constraint node(s) as shown in Fig. 7. Intermediate tables are necessary to make connections among tables for answer node(s) and tables for constraint node(s). Finally, in a *WHERE* clause, we explicitly make connections for tables in the *FROM* clause and set values (i.e., names of constraint nodes) to retrieve appropriate records in tables. The number of connections we need to make is one less than the number of tables in the *FROM* clause. Connections are made through sharing attributes (e.g., *author_id* between *AuthorInfo* and *OrganizationInfo*). Names of the constraint node(s) in a graph query is set to name attributes (e.g., *paper_name* for the *Paper* table) of non-intermediate tables.

*4.4.2. The translation of graph queries into Cypher commands*

We show how graph queries can be translated into Cypher commands that can query data stored in Neo4j (a graph database), which is implemented based on the property graph model shown in Fig. 1. Unlike SQL in relational databases, there are many query languages available in graph databases. Even though these query languages use different syntaxes, they share the same components and can be translated into each other easily. Fig. 8 shows the same four example graph queries shown in Fig. 7 and their translations into Cypher commands. The translation of graph queries to Cypher commands is simple, because Cypher is itself a query language based on graph patterns. Other graph database query languages such as SPARQL and Gremlin have similar syntaxes with Cypher, and the general constructs are primarily the same. Readers can refer to Holzschuher and Peinl's work (2013) for a comparison of different graph query languages.

The three main constructs of Cypher are *MATCH*, *WHERE*, and *RETURN*. We specify graph patterns in a *MATCH* clause and set constraints to nodes in a *WHERE* clause. A *RETURN* clause is for the data that we want to retrieve. A *RETURN* clause in Cypher corresponds to a *SELECT* clause in SQL, and both are for specifying data need to be returned by the database. The *WHERE* clause in Cypher is much simpler than the one in SQL. We only need to set values to the bibliographic entities that are designated as constraint nodes when a user formulating a graph query. Constructing a *MATCH* clause is straightforward, which is a textual representation of a graph query, with the same nodes and relations with the same directions. The only thing we need to do is to set an arbitrary and unique instance name for each node. Instance names can be simply strings

| Visual Graph Query | SELECT | FROM | WHERE |
|---|---|---|---|
|  | Paper.paper_name | Paper,<br>AuthorInfo,<br>Author | Paper.paper_id = AuthorInfo.paper_id AND<br>AuthorInfo.author_id = Author.author_id AND<br>Author.author_name = "John" |
|  | Source.source_name | Source,<br>SourceInfo,<br>AuthorInfo,<br>Author | Source.source_id = SourceInfo.source_id AND<br>SourceInfo.paper_id = AuthorInfo.paper_id AND<br>AuthorInfo.author_id = Author.author_id AND<br>Author.author_name = "John" |
|  | Organization.organization_name | Organization,<br>OrganizationInfo,<br>AuthorInfo,<br>ReferenceInfo,<br>Paper | Organization.organization_id = OrganizationInfo.organization_id AND<br>OrganizationInfo.author_id = AuthorInfo.author_id AND<br>AuthorInfo.paper_id = ReferenceInfo.cited_paper_id AND<br>ReferenceInfo.citing_paper_id = Paper.paper_id AND<br>Paper.paper_name = "Introduction to Database Systems" |
|  | Author.author_name | Author,<br>AuthorInfo,<br>ReferenceInfo,<br>SourceInfo,<br>Source,<br>TermInfo,<br>Term | Term.term_id = TermInfo.term_id AND<br>TermInfo.paper_id = ReferenceInfo.citing_paper_id AND<br>Source.source_id = SourceInfo.source_id AND<br>SourceInfo.paper_id = ReferenceInfo.citing_paper_id AND<br>ReferenceInfo.cited_paper_id = AuthorInfo.paper_id AND<br>AuthorInfo.author_id = Author.author_id AND<br>Term.term_name = "Information Retrieval" AND<br>Source.source_name = "SIGIR" |

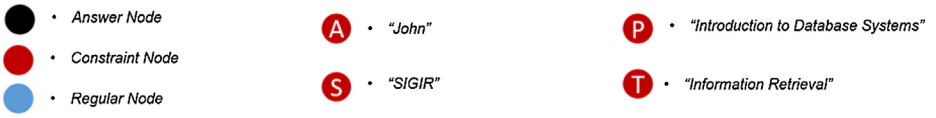
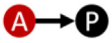
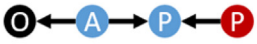
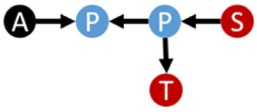



**Fig. 7.** Examples of translating visual graph queries into SQL.

| Visual Graph Query | MATCH | WHERE | RETURN |
|---|---|---|---|
|  | (a:Author)-->(p:Paper) | a.name = 'John' | p.name |
|  | (a:Author)-->(p:Paper),<br>(p:Paper)<--(s:Source) | a.name = 'John' | s.name |
|  | (o:Organization)<--(a:Author),<br>(a:Author)-->(cited_p:Paper),<br>(cited_p:Paper)<--(citing_p:Paper) | citing_p.name = 'Introduction to Database Systems' | o.name |
|  | (a:Author)-->(cited_p:Paper),<br>(cited_p:Paper)<--(citing_p:Paper),<br>(citing_p:Paper)<--(s:Source),<br>(citing_p:Paper)-->(t:Term) | t.name = "Information Retrieval' AND<br>s.name = "SIGIR' | a.name |

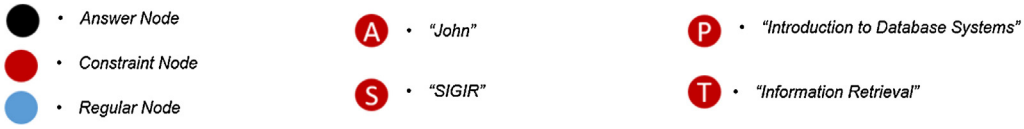



**Fig. 8.** Examples of translating visual graph queries into Cypher.

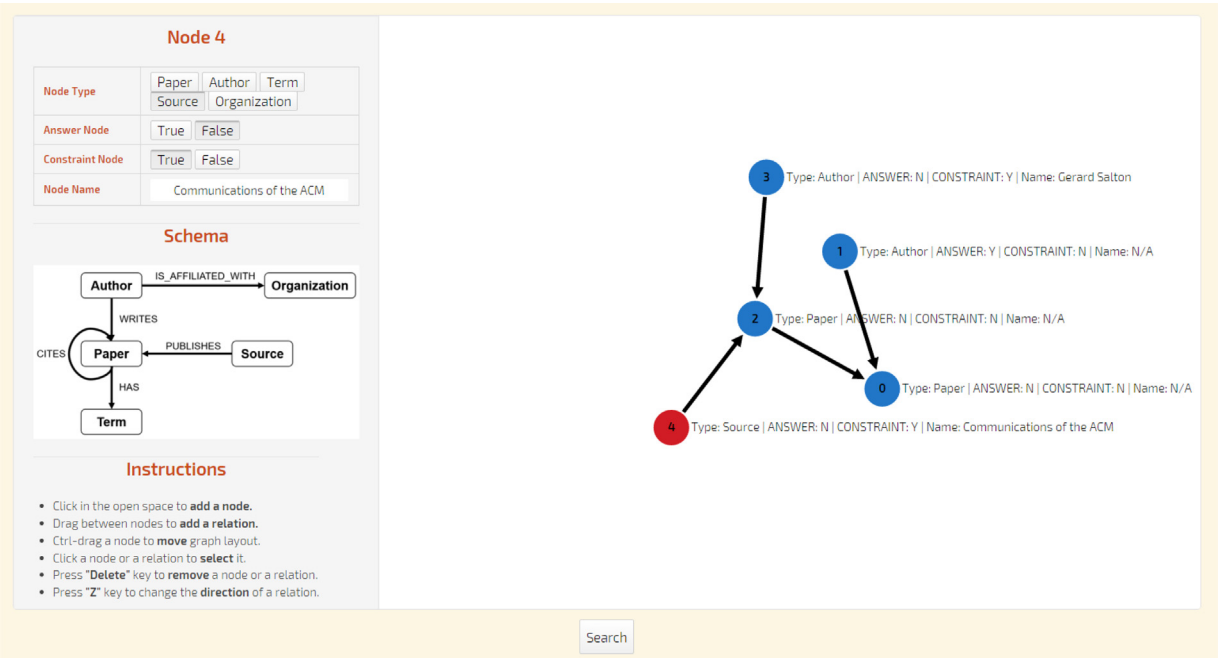



**Fig. 9.** The visual graph query interface of the system.

with/without numbers that can differentiate nodes from each other. An instance name of a node (e.g., *cited_p*) is accompanied by the node type (e.g., *Paper*), which guides the traversal of the graph.

As previously shown, graph queries can be translated into relational and graph database queries as long as a data schema is available. Thus, existing bibliographic retrieval systems that use relational databases can adopt a visual graph query interface and new systems can adopt graph databases for easy system implementation and better performance in terms of retrieval speed.

## 5. Results

### 5.1. The designed system

We implemented a web-based bibliographic information retrieval system with a visual graph interface. The system is based on the Spring Framework and the interface (i.e., front end) is implemented using JavaScript libraries (jQuery and D3.js). We used Neo4 j (a graph database) to build the database layer. The example dataset used in the system is provided by Tang et al. (2008), which contains 629,814 papers, 595,775 authors, 12,609 sources, 291,109 terms, and 1000 organizations. Detailed descriptions on the transformation of the original dataset can be found in our previous work (Zhu et al., 2016). In the following paragraphs, we show the implemented system with example queries. Fig. 9 shows the visual graph query interface of the system.

The visual graph query interface is composed of two panels: the configuration panel (left) and the graph query panel (right). The configuration panel has three sections: a node configuration section, a schema section, and an instructions section. As shown in the instructions section, users can create a node, drag between nodes to add a link, and change the direction of the link by pressing the Z key. Deleting a node or a relation can be done by pressing the Delete key after selecting a node or a relation. The visual graph query shown in Fig. 9 has five nodes and each node has an id from zero to four. Attributes are shown next to the nodes; for example, *node 1* has a string value of "Type: Author | ANSWER: Y | CONSTRAINT: N | Name: N/A". It means that the type of *node 1* is *Author*, and it is an answer node. In addition, the node is not a constraint node and does not have a name. Users can select a node (e.g., *node 4*) to set and change attributes of the node. *Node 4* is a constraint node and it has the name of "*Communications of the ACM*". The visual graph query shown in Fig. 9 denotes "*authors of papers that were cited by papers that were written by Gerard Salton and published in Communications of the ACM*". By clicking the search button, candidate graph queries are shown (Fig. 10(a)).

Only one candidate graph query is shown because the formulated graph query is syntactically and semantically correct. As shown in Fig. 10, all links are added with labels and each node is filled with appropriate colors (i.e., black for the answer node, red for constraint nodes, and blue for regular nodes). Search results of the candidate graph query is shown in Fig. 10(b).

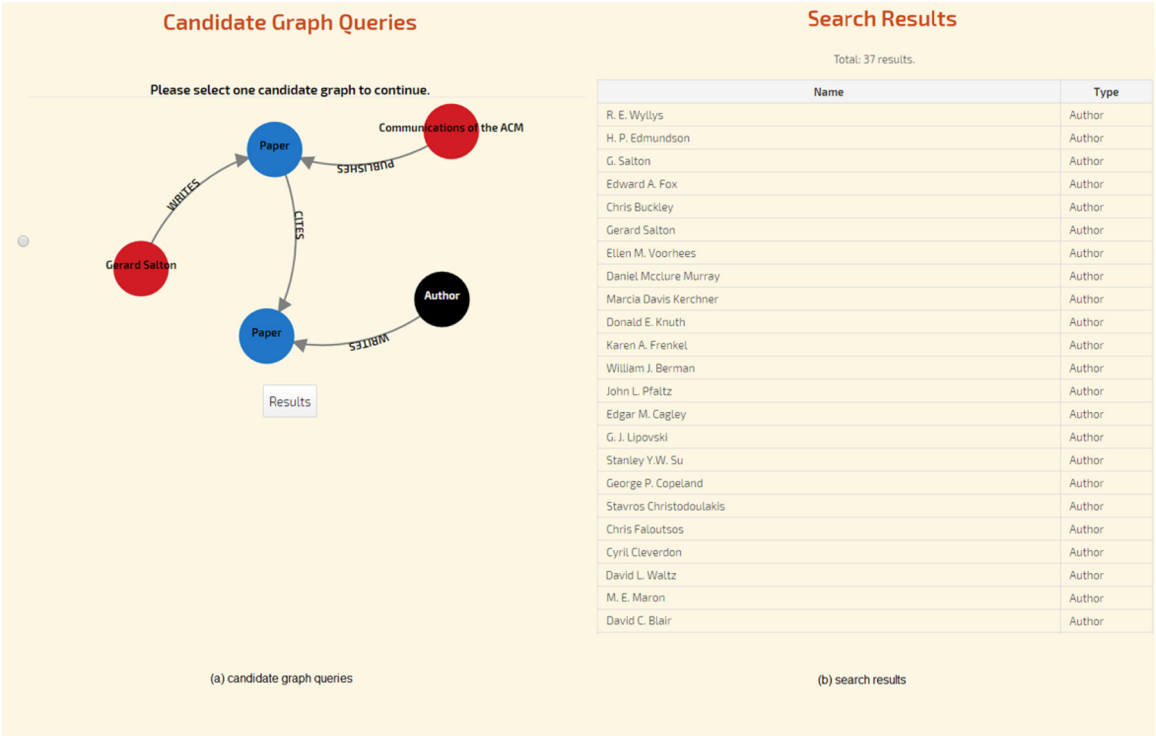



**Fig. 10.** Candidate graph queries and search results.

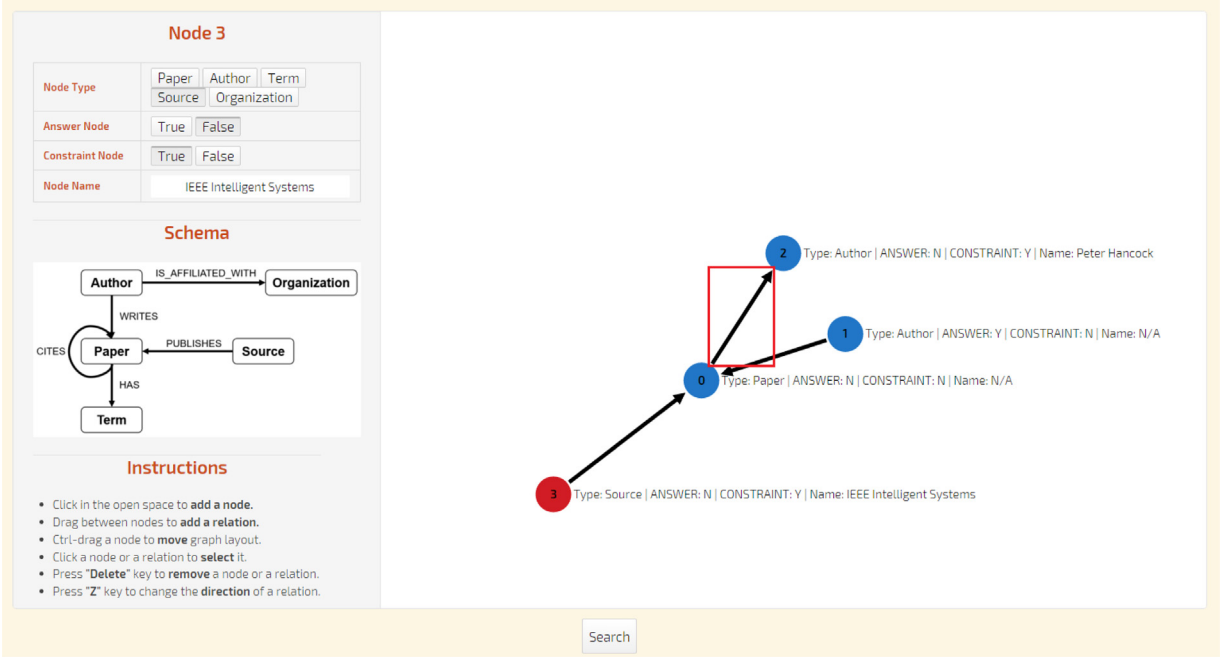



**Fig. 11.** A semantically incorrect visual graph query.

Next, we show how a semantically incorrect visual graph query is corrected and disambiguated. Fig. 11 shows an example query, in which the link from *node 0* to *node 2* is incorrect. The direction of the link is opposite to the schema, and thus needs to be further analyzed to identify all possible interpretations.

As shown in Fig. 12, two candidate graph queries are formulated from the above visual graph query for users to select. The visual graph query can be interpreted as either "*authors of papers that were written by Peter Hancock and published in*

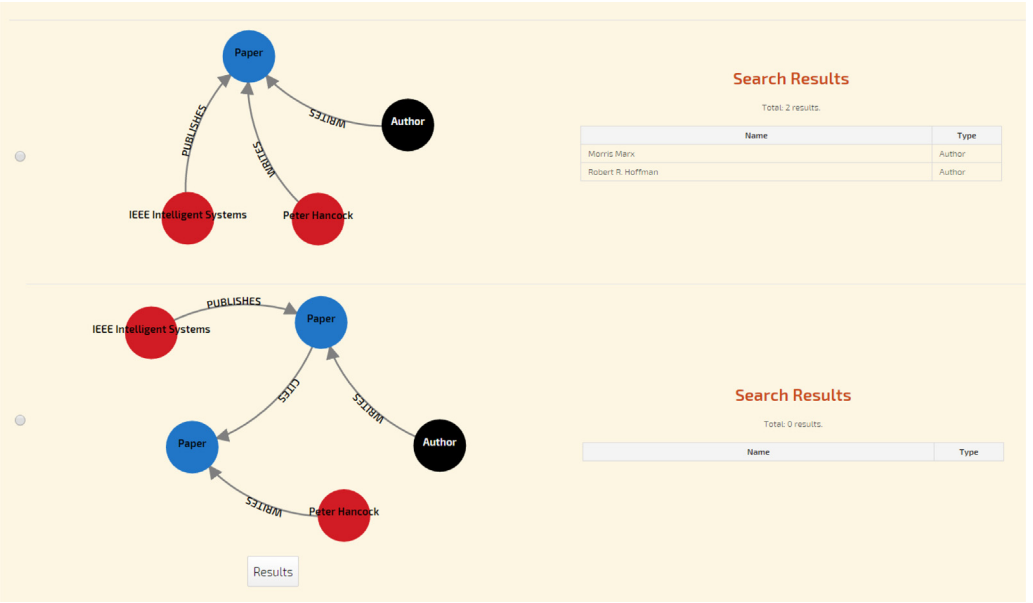



**Fig. 12.** Two candidate graph queries and search of the query in Fig. 11.

*IEEE Intelligent Systems*" (for finding co-authors of *Peter Hancock*) or "*authors of papers that were published in IEEE Intelligent Systems and cited papers that were written by Peter Hancock*". The two candidate graph queries resulted in different search results.

### 5.2. Comparison of the designed system with current bibliographic IR systems

In this section, we evaluate the visual graph query interface by comparing the designed system with current bibliographic IR systems in terms of functionality and expressive power. Functionality denotes any functional features of bibliographic information retrieval systems that facilitate the retrieval of bibliographic information. It is measured by investigating any functional strengths of our system against the existing systems. Expressive power is operationalized as a system's ability of representing bibliographic information needs with bibliographic queries and answering those queries. It is measured by the ratio of directly representable queries in each system.

#### 5.2.1. The support of entity types

Current bibliographic IR systems typically only provide papers as the final search results. This means we cannot get other entity types (e.g., authors) in search results without using additional add-on features (e.g., "Analyze Results" in Web of Science). The designed system supports search for multiple entity types including papers, authors, terms, sources, and organizations. For example, the example query shown in Fig. 9 is formulated to retrieve authors. This allows the retrieval of search results within a single step and reduces the cost of going through multiple post-search refining steps.

Another strength of our system is that it supports retrieving multiple types of entities in one search. Fig. 13 shows an example query that has two answer nodes (*node 1* and *node 2*).

The natural language representation of the query is "*authors and sources of papers that were cited by papers written by Thomas Heinis*". As shown in Fig. B1, the search results included two types of bibliographic entities *Author* and *Source*. This function reduces the number of queries required to complete a given search task that might need multiple queries to satisfy the information need.

#### 5.2.2. The representability of real-world bibliographic queries

Due to the lack of readily available complied resources, twenty test queries were handcrafted by the authors. The twenty test queries were constructed and divided into four groups based on the number of bibliographic nodes (i.e., from two to five) in a query. Because creating queries is an open problem, we were not able to apply standard measures such as inter-coder reliability, but performed several rounds of discussions to exclude subjectivity as much as possible. As a follow-up examination, we consulted three researchers to review the test queries. Researchers were asked to rank those queries that best match their everyday bibliographic information needs, and recommend additional queries that were not included in the original test queries. We selected top 10 queries ranked by the researchers (referred to as regular queries in Table 1). In addition, we included five advanced queries of common bibliometric tasks such as bibliographic coupling, paper co-citation, author co-citation, co-author, and co-word. The final 15 test queries are shown in Appendix C. We benchmarked

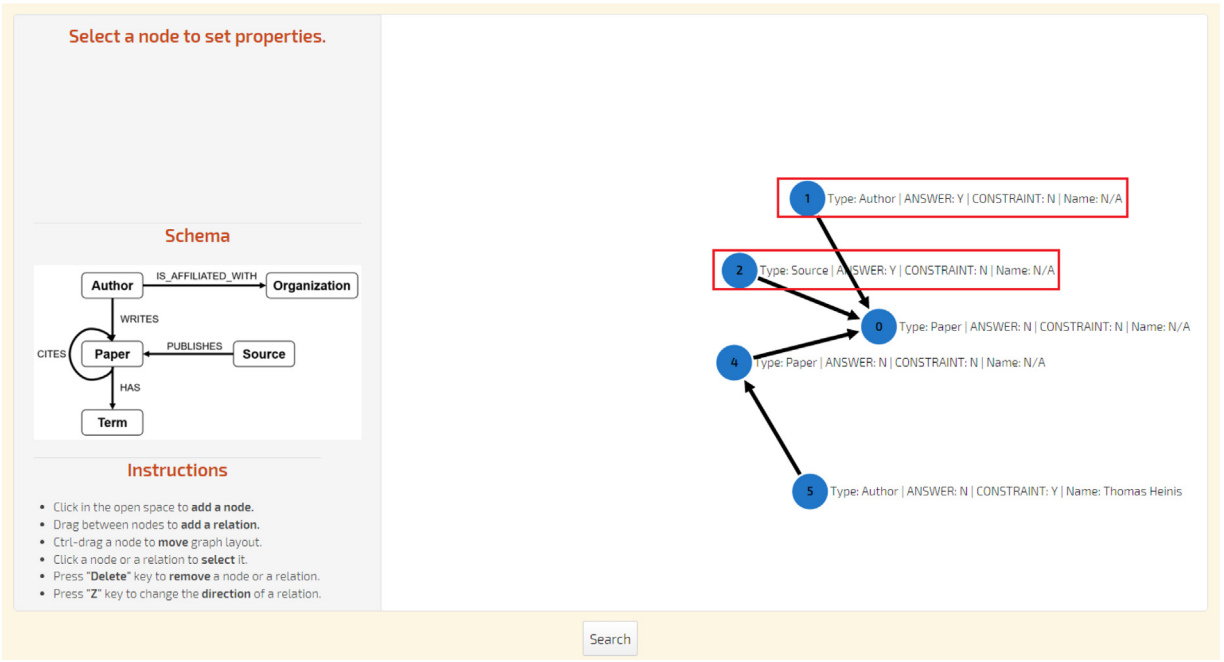



**Fig. 13.** An example query with two answer nodes.

**Table 1**
The ratio of directly representable queries in each system.

| Test queries | | Web of Science | Scopus | Google Scholar | Our system |
|---|---|---|---|---|---|
| Regular queries | | 7/10 | 7/10 | 3/10 | 10/10 |
| (unanswerable queries) | | (No. 4, 5, & 10) | (No. 4, 5, & 10) | (No. 3, 4, 5,7,8, 9, and 10) | (None) |
| Advanced queries | Bibliographic coupling | N | N | N | Y |
| | Paper co-citation | N | N | N | Y |
| | Author co-citation | N | N | N | Y |
| | Co-author | Y | Y | Y | Y |
| | Co-word | Y | Y | N | Y |

our system's representability of these queries in relation to three major bibliographic information retrieval systems (i.e., Web of Science, Scopus, and Google Scholar). We measure the extent to which these queries are representable in one query and directly answerable without initiating other search tasks. For fair comparisons, we included the add-on features of the major systems (e.g., "Analyze Results" of Web of Science) and treated them as integrated components of the search tasks. Table 1 shows the ratio of queries that can be represented in each system with unanswerable queries listed in parentheses.

As shown in Table 1, existing systems have limitations in directly representing the information included in the test queries, despite the fact that we took their additional add-on features into consideration. Google Scholar did not fulfill majority of the tasks because it is not designed to retrieve entity types other than papers and does not explicitly manage metadata. Web of Science and Scopus performed better than Google Scholar, but were not able to represent a few test queries directly (i.e., bibliographic coupling, paper co-citation, and author co-citation). We were able to retrieve entity types other than papers by using their add-on features (e.g., "Analyze Results") in Web of Science and Scopus, but the two systems do not support a feature where users can conduct a follow-up search through the add-on options. For example, items (e.g., authors) displayed in the "*Analyze Results*" feature are only used to refine the previous search results. Using the fourth test query as an example ("Papers of authors who wrote *Term-weighting Approaches in Automatic Text Retrieval*"), a general solution to this query involves getting author names of the paper and retrieving papers written by those papers. Web of Science supports getting author names by providing the title of the paper; however, it does not support the retrieval of papers written by those papers (though one can manually record the author names and initiate another round of search). For the advanced queries, the strength of our system is more evident. Our system was able to represent all the queries that are common for bibliometric tasks.

## 6. Discussion

While the proposed visual graph query interface is not specifically designed for a certain group of users, as shown in the previous section, it will be beneficial to bibliometricians and researchers who have complex bibliographic information needs.

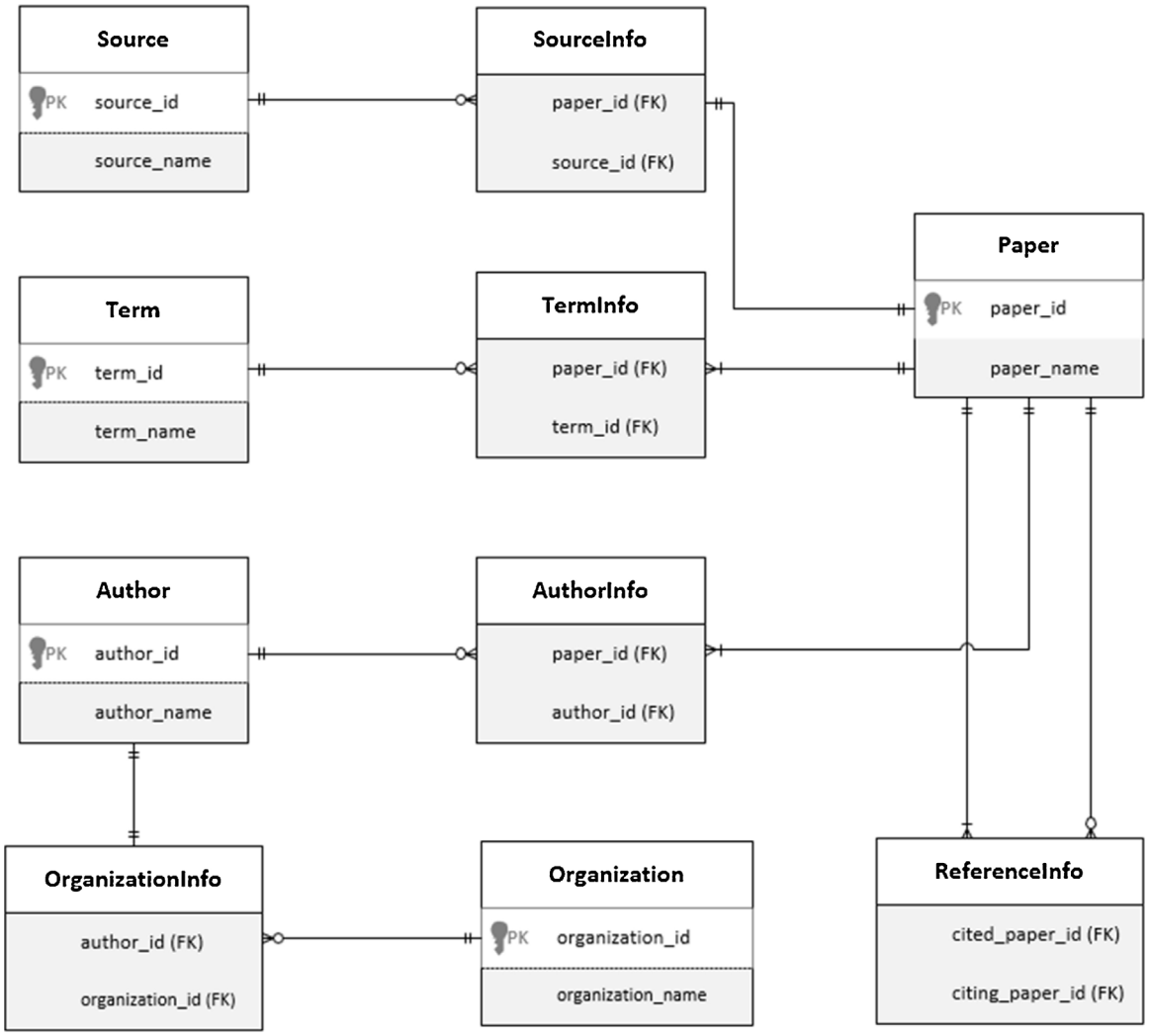



**Fig. A1.** An entity-relationship model for bibliographic data. Tables for five bibliographic entities, five intermediate tables (i.e., ReferenceInfo, SourceInfo, TermInfo, AuthorInfo, and OrganizationInfo), and relations with appropriate cardinalities are shown. For example, one author can write zero, one, or many papers, and one paper can have only one source (i.e., one journal or one conference).

Simple bibliographic queries might be easily formulated in the form-based systems. For instance, common bibliometric tasks such as bibliographic coupling and author co-citation can be succinctly accomplished by the proposed interface while simple queries such as "*papers written by Gerard Salton*" are readily answerable by form-based systems. In this regard, the proposed interface is complementary to form-based systems and is tailored more towards intermediate or advanced users. Thus, there may be a trade-off between the expressive power and usability. Advanced users who have the knowledge of directly constructing database query languages (e.g., SQL and SPARQL) can benefit from the interface by formulating graph queries. Due to the scope of the study, ways of presenting search results were not studied in detail. This might include issues of sorting search results and presenting multiple types bibliographic entities in a single search result. Future work is planned to study these problems and provide appropriate solutions.

Our schema (Fig. 1) does not include all constraints (or properties) of bibliographic entities, which might limit its application. As a future work, we plan to consider additional constrains to enable the formulation of a variety of bibliometric queries. Because of the scarcity of research in visual graph queries, we were unable to compare the results of our methods with a baseline. In addition, due to different computational environment (e.g., server capacity) of our system and the existing bibliographic information retrieval systems, quantitative evaluation (e.g., query processing time) is not applicable. Therefore, the evaluations of the visual graph query interfaces were largely focused on its functionality and expressive power. As a future work, we plan to perform user-centric evaluations to understand the usability aspect of the visual graph query interfaces.

## Search Results

Total: 44 results.

| Name | Type |
|---|---|
| Anish Das Sarma | Author |
| Proceedings of the 32nd international conference on Very large data bases | Source |
| Chris Hayworth | Author |
| Parag Agrawal | Author |
| Omar Benjelloun | Author |
| Jennifer Widom | Author |
| Shubha Nabar | Author |
| Tomoe Sugihara | Author |
| John Darrell Van Horn | Author |
| Journal of Cognitive Neuroscience | Source |
| Wang Chiew Tan | Author |
| Proceedings of the 8th International Conference on Database Theory | Source |
| Sanjeev Khanna | Author |
| Peter Buneman | Author |
| Wahid Chrabakh | Author |
| Proceedings of the 2003 ACM/IEEE conference on Supercomputing | Source |
| Rich Wolski | Author |
| Rajendra Bose | Author |
| ACM Computing Surveys (CSUR) | Source |
| James Frew | Author |
| Gustavo Alonso | Author |
| Proceedings of the 17th International Conference on Data Engineering | Source |
| Cesare Pautasso | Author |
| Win Bausch | Author |

**Fig. B1.** Search results of the query shown in Fig. 13. The search results include both authors and sources.

## 7. Conclusion

In this paper, we proposed a visual graph query interface for bibliographic information retrieval. We first introduced a visual graph query interface, through which, users can formulate bibliographic queries by drawing nodes and links. We introduced methods of interpreting and translating graph queries into relational and graph database queries and implemented a web-based bibliographic information retrieval system with a visual graph query interface. We designed and achieved several novel procedures such as query correction, query disambiguation, and candidate graph query selection and they exhibited value in real-world use cases. Based on a comparison with an existing bibliographic information retrieval system, we examined the strength of the visual graph query interface in representing complex queries. The visual graph query interface outperformed the compared system in the aspect of representable queries. While the major systems failed to represent certain queries, the visual graph query interface was able to represent all test queries of varying lengths. In addition,

the visual graph query showed two advanced features that are not attainable in the benchmarked systems: the support of multiple entity types as the final search results and the support of more than one entity type in each search.

In the big data era, we have witnessed the dramatic growth of bibliographic data. In order to get more relevant search results, the representation of more precise information needs is vital. Queries representing explicit information needs can dramatically reduce the size of search results and minimize refining operations. The visual graph query interface designed in this study reduces multiple searching and refining steps. In this regard, visual graph query interfaces are a practical solution for searching and retrieving bibliographic data.

## Author contributions

Conceived and designed the analysis: Yongjun Zhu and Erjia Yan.
Collected the data: Yongjun Zhu and Erjia Yan.
Contributed data or analysis tools: Yongjun Zhu and Erjia Yan.
Performed the analysis: Yongjun Zhu and Erjia Yan.
Wrote the paper: Yongjun Zhu and Erjia Yan.
Other contribution: Yongjun Zhu and Erjia Yan.

## Acknowledgements

This project was made possible in part by the Institute of Museum and Library Services (Grant Award Number: RE-07-15-0060-15), for the project titled "Building an entity-based research framework to enhance digital services on knowledge discovery and delivery".

## Appendix A. : Database schemas: An ER model and a property graph model

## Appendix B. : Experimental results

## Appendix C. : Twenty queries tested in the experiment

1.) Regular queries
2.) Papers written by Gerard Salton
3.) Papers on the topic of Human-Computer Interaction
4.) Papers that were cited by "Introduction to Modern Information Retrieval"
5.) Papers of authors who wrote "Term-weighting Approaches in Automatic Text Retrieval"
6.) Papers on the topics of "The Hadoop distributed file system"
7.) Papers that were written by Christopher D Manning and published by Association for Computational Linguistics
8.) Authors at CMU, who presented papers in SIGCHI
9.) Authors who wrote papers on the topic of information seeking behavior, which were presented in SIGIR
10.) Topics of papers that were presented in SIGKDD
11.) Conferences that presented papers on the topics discussed in "MapReduce: simplified data processing on large clusters"

Advanced queries

1.) Bibliographic coupling (papers that were cited by "MapReduce: simplified data processing on large clusters" and "The Hadoop distributed file system")
2.) Paper co-citation (papers that cited both MapReduce: simplified data processing on large clusters" and "The Hadoop distributed file system")
3.) Author co-citation (papers that cited both Gerard Salton and James Allan)
4.) Co-author (authors who co-authored with Gerard Salton)
5.) Co-word (keywords that co-occurred with big data)

## References

Aggarwal, C. C., & Wang, H. (2010). *Managing and mining graph data* (1st ed.). New York: Springer. http://dx.doi.org/10.1007/978-1-4419-6045-0
Baeza-Yates, R., Brisaboa, N., & Larriba-Pey, J. (2010). *A model for automatic generation of multi-partite graphs from arbitrary data.* pp. 49–60. Berlin, Heidelberg: Springer. http://dx.doi.org/10.1007/978-3-642-16720-1_5
Ceri, S., Comai, S., Damiani, E., & Fraternali, P. (1999). XML-GL: A graphical language for querying and restructuring XML documents. *Computer Networks, 31*(11–16), 1171.
Cook, D. J., & Holder, L. B. (Eds.). (2006). Mining graph data. In. John Wiley & Sons.
Erwig, M. (2003). Xing: A visual XML query language. *Journal of Visual Languages and Computing, 14*(1), 5–45. http://dx.doi.org/10.1016/S1045-926X(02)00074-5

Gómez-Villamor, S., Soldevila-Miranda, G., Giménez-Vañó, A., Martínez-Bazan, N., Muntés-Mulero, V., & Larriba-Pey, J. (2008). BIBEX: A bibliographic exploration tool based on the DEX graph query engine. *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, 735–739. http://dx.doi.org/10.1145/1353343.1353439

Giugno, R., & Shasha, D. (2002). Graphgrep: A fast and universal method for querying graphs. *16th international conference on pattern recognition, 2002* (vol. 2) (pp. 112–115).

Han, L., Finin, T., & Joshi, A. (2012). GoRelations: an intuitive query system for DBpedia. In *The semantic web*. pp. 334–341. Berlin, Heidelberg: Springer.

He, H., & Singh, A. K. (2008). Graphs-at-a-time: Query language and access methods for graph databases. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (pp. 405–418).

Hearst, M. (2009). *Search user interfaces*. Cambridge University Press.

Hogenboom, F., Milea, V., Frasincar, F., & Kaymak, U. (2010). RDF-GL: A SPARQL-based graphical query language for RDF. In *Emergent web intelligence: advanced information retrieval*. pp. 87–116. London: Springer.

Holzschuher, F., & Peinl, R. (2013). Performance of graph query languages: Comparison of cypher, gremlin and native access in neo4j. In *Proceedings of the joint EDBT/ICDT 2013 workshops* (pp. 195–204).

Jayaram, N., Khan, A., Li, C., Yan, X., & Elmasri, R. (2015). Querying knowledge graphs by example entity tuples. *IEEE Transactions on Knowledge and Data Engineering*, *27*(10), 2797–2811. http://dx.doi.org/10.1109/TKDE.2015.2426696

Jiang, H., Wang, H., Yu, P. S., & Zhou, S. (2007). Gstring: A novel approach for efficient search in graph databases. In *IEEE 23rd international conference on data engineering, 2007. ICDE 2007* (pp. 566–575).

Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., & Karambelkar, H. (2005). Bidirectional expansion for keyword search on graph databases. In *Proceedings of the 31st international conference on very large data bases* (pp. 505–516).

Leser, U. (2005). A query language for biological networks. *Bioinformatics*, *21*(S2), ii33–ii39. http://dx.doi.org/10.1093/bioinformatics/bti1105

Ni, W., & Ling, T. W. (2003). GLASS: A graphical query language for semi-structured data. In *Eighth international conference on database systems for advanced applications. (DASFAA 2003)* (pp. 363–370).

Pradel, C. (2012). Allowing end users to query graph-based knowledge bases. In *Knowledge engineering and knowledge management*. pp. 8–15. Berlin, Heidelberg: Springer.

Schweiger, D., Trajanoski, Z., & Pabinger, S. (2014). SPARQLGraph: A web-based platform for graphically querying biological semantic web databases. *BMC Bioinformatics*, *15*(1), 279. http://dx.doi.org/10.1186/1471-2105-15-279

Sun, Y., Barber, R., Gupta, M., Aggarwal, C. C., & Han, J. (2011). Co-author relationship prediction in heterogeneous bibliographic networks. In *2011 international conference on advances in social networks analysis and mining (ASONAM)* (pp. 121–128).

Tang, J., Zhang, J., Yao, L., Li, J., Zhang, L., & Su, Z. (2008). Arnetminer: Extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 990–998).

Tran, T., Wang, H., Rudolph, S., & Cimiano, P. (2009). Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In *25th international conference on data engineering, 2009. ICDE'09* (pp. 405–416).

Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., & Wilkins, D. (2010). A comparison of a graph database and a relational database: A data provenance perspective. In *Proceedings of the 48th annual Southeast regional conference* [p. 42].

Yan, E., & Ding, Y. (2009). Applying centrality measures to impact analysis: A coauthorship network analysis. *Journal of the American Society for Information Science and Technology*, *60*(10), 2107–2118.

Ykhlef, M., & Alqahtani, S. (2009). GQLX: A new graphical query language for XML data. *Proceedings of the 11th international conference on information integration and web-based applications & services*, 201–208. http://dx.doi.org/10.1145/1806338.1806379

Ykhlef, M., & Alqahtani, S. (2011). A survey of graphical query languages for XML data. *Journal of King Saud University – Computer and Information Sciences*, *23*(2), 59–70. http://dx.doi.org/10.1016/j.jksuci.2011.05.002

Yuan, Y., Wang, G., Chen, L., & Wang, H. (2013). Efficient keyword search on uncertain graph data. *IEEE Transactions on Knowledge and Data Engineering*, *25*(12), 2767–2779.

Zhang, Y., Zhang, N., Tang, J., Rao, J., & Tang, W. (2010). Mquery: Fast graph query via semantic indexing for mobile context. *IEEE/WIC/ACM international conference on web intelligence and intelligent agent technology (WI-IAT), 2010* (vol. 1) (pp. 508–515).

Zhou, Q., Wang, C., Xiong, M., Wang, H., & Yu, Y. (2008). *SPARK: Adapting keyword query to semantic search*. pp. 694–707. Berlin, Heidelberg: Springer. http://dx.doi.org/10.1007/978-3-540-76298-0_50

Zhu, Y., Yan, E., & Song, I.-Y. (2016). The use of a graph-based system to improve bibliographic information retrieval: System design, implementation, and evaluation. *Journal of the Association for Information Science and Technology*, http://dx.doi.org/10.1002/asi.23677