BiQL: a query language for analyzing information networks

Anton Dries[1,2], Siegfried Nijssen[1] and Luc De Raedt[1]

[1] Katholieke Universiteit Leuven, Belgium
[2] Universitat Pompeu Fabra, Barcelona, Spain

Abstract. One of the key steps in data analysis is the exploration of data. For traditional relational data, this process is facilitated by relational database management systems and the aggregates and rankings they can compute. However, for the exploration of graph data, relational databases may not be most practical and scalable. Many tasks related to exploration of information networks involve computation and analysis of connections (e.g. paths) between concepts. Traditional relational databases offer no specific support for performing such tasks. For instance, a statistic such as the shortest path between two given nodes cannot be computed by a relational database. Surprisingly, tools for querying graph and network databases are much less well developed than for relational data, and only recently an increasing number of studies are devoted to graph or network databases. Our position is that the development of such graph databases is important both to make basic graph mining easier and to prepare data for more complex types of analysis.
In this chapter, we present the BiQL data model for representing and manipulating information networks. The BiQL data model consists of two parts: a data model describing objects, link, domains and networks, and a query language describing basic network manipulations. The main focus here lies on data preparation and data analysis, and less on data mining or knowledge discovery tasks directly.

1 Introduction

Information networks are a popular way of representing information. In its most basic form, such a network can be seen as a set of objects, interconnected by links. Because of this link structure, these networks are capable of representing complex information using a simple data model. Information networks can be found in a wide variety of domains, for example, as social networks, bibliographical networks, and biological networks such as gene-protein interaction networks and pathways. Although all these examples seem very different, their analysis requires many similar operations. For example, determining the influence of a publication in a citation network is similar to finding the role of a gene in a biological pathway, finding the well-connected users in a social network corresponds to finding the important traffic hubs in a road network, and network analysis algorithms such as PageRank can be applied to different types of networks such as the world wide web and social networks. Because of this common structure it seems natural to look for a common infrastructure to deal with these networks.

Currently, different graph databases are available (e.g. DEX [31] and Neo4j [32]). However, many of these systems focus mainly on low-level aspects such as data structures and algorithms, instead of higher level concepts such as providing a simple data model and query language. In this article, we take a different approach and we focus on developing a data model for information networks that is suitable for network analysis and data mining. This data model, called BiQL (or **Bi**son **Q**uery **L**anguage), aims at providing a powerful set of operations for manipulating a wide variety of heterogeneous networks. Within the knowledge discovery process, BiQL mainly focusses on preprocessing, transformation, analysis, and, to a lesser extent, data mining.

In this chapter, we give a general overview of the BiQL system. For a more in-depth discussion on the query language and its underlying operations we refer the reader to [19, chapter 6].

2 Motivating example

Consider the bibliographic network shown in Figure 1. This network contains authors, publications, keywords, citations, authorship and keyword relations.

Such a network can be used and analyzed in many ways. For example, one could be interested in doing co-authorship analysis. In that case the 'publication' nodes are considered to be edges between 'authors' and the network can be represented as shown in Figure 2. The co-author relationship can be expressed using regular edges (Figure 2a) or using hyperedges (Figure 2b).

Alternatively, one may be interested in analyzing publications for each domain separately by splitting up the network into a set of networks, one for each keyword, as can be seen in Figure 3.

Many more cases can be imagined, for example, citation analysis between publications, authors, or even keyword domains. In order to be able to perform all these tasks, we need a data representation and query language that are capable of representing, manipulating and transforming information networks. Moreover, we also want to analyse such networks, that is, calculate aggregate measures, apply ranking functions, and store the results back in the network for future querying. In general, we can identify a number of key tasks that a network management system should support:

1. *Introduce new relationships in the network*, for example, create a 'co-author' relationship between authors that have published a paper together, or create a citation relation between authors based on the citation relation between publications.
2. *Find connections between objects*, for example, find co-citations between authors, that is, author A cites author B and author B cites author A (possibly indirectly).

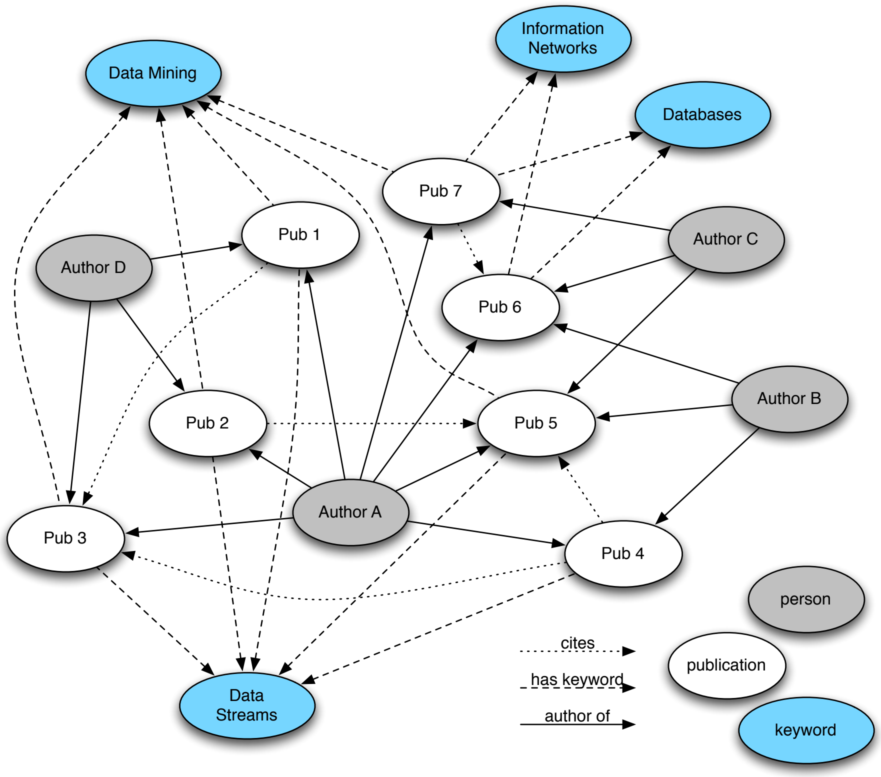
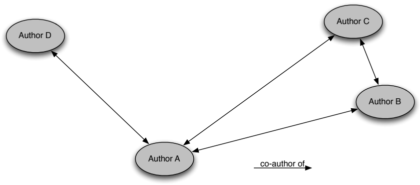
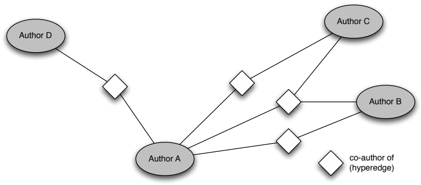


Fig. 1: Bibliographic network containing the entities ‘authors’, ‘publications’ and ‘keywords’, and the ‘author of’, ‘has keyword’, and ‘cites’ relationships.

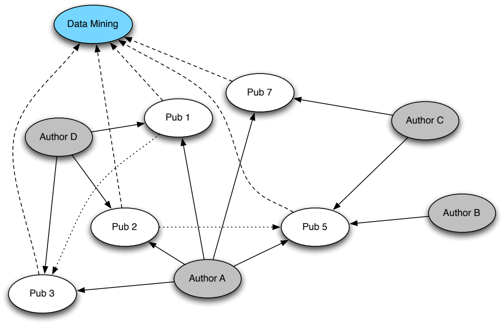


(a) Using regular edges

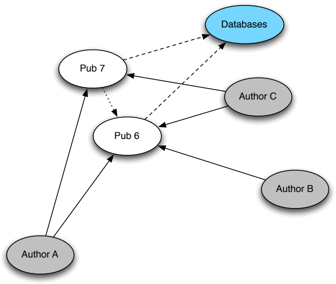


(b) Using hyperedges

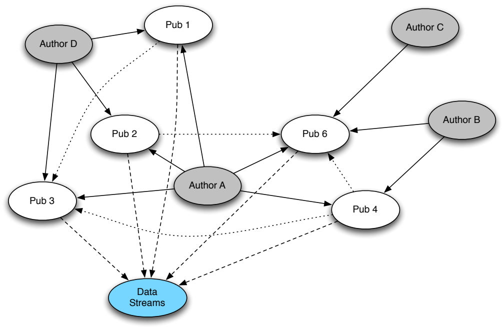
Fig. 2: Network from Figure 1 transformed for co-authorship analysis.



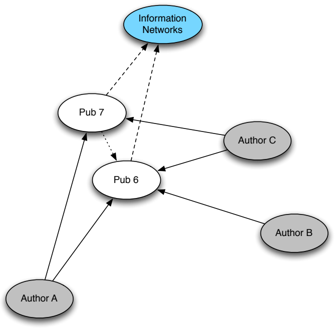
(a) "Data Mining"



(b) "Databases"



(c) "Data Streams"



(d) "Information Networks"

Fig. 3: Network from Figure 1 separated by keyword.

3. *Find the transitive closure of a relation*, for example, find the influence graph of a publication based on citations, or the co-author neighbourhood of an author.
4. *Rank results*, for example, find the authors with the most co-authors, or with the largest co-author network.
5. *Calculate network analysis metrics*, for example, centrality of an author in the co-author network.
6. *Introduce weights and probabilities*, and use them in probabilistic queries.
7. *Discover bisociations or other non-obvious connections*, for example, by comparing different distance measures.
8. *Apply external algorithms on the network*, for example, for finding quasi-cliques [42].

Our goal is to support all these tasks.

3 Requirements

The main motivation and target application for our data model and query language is supporting exploratory data analysis on networked data, which means our system is intended to be part of the knowledge discovery process. This results in the following requirements and design choices.

Small is beautiful The data model should consist of a small number of concepts and primitives. As a consequence, we do not wish to introduce special language constructs to deal with complicated types of networks (directed, undirected, labeled, hypergraphs, etc.) or sets of graphs.

Uniform representation of nodes and edges The most immediate consequence of the former choice is that we wish edges and nodes to be represented in a uniform way. We will do this by representing both edges and nodes as objects that are linked together by links that have no specific semantics. This also allows one to generate different views on a network. For instance, in a bibliographic database, we may have objects such as papers, authors and citations. In one context one could analyze the co-author relationship, in which case the authors are viewed as nodes and the papers as edges, while in another context, one could be more interested in citation-analysis, in which case the papers are the nodes and the citations the edges.

Closure property The result of any operation or query can be used as the starting point for further queries and operations. The information created by a query combined with the original database can therefore be queried again.

SQL-based There are many possible languages that could be taken as starting point, such as SQL, relational algebra or Datalog. We aimed for a data model on which multiple equivalent ways to represent queries can be envisioned. The queries that we propose on this model are expressed in an SQL-like notation here, as this notation is more familiar to many users of databases, and is the prime example of a declarative query language.

Aggregates To support a basic analysis of graphs, we need to be able to calculate statistics such as

- the degree of nodes;
- the number of nodes reachable from a certain node (connected component size);
- the length of a shortest path between two nodes;
- the length of the longest shortest path from one node to all other nodes (closeness centrality);
- the sum or product of weights on edges on paths.

These statistics are not only useful when obtaining an initial insight in data. It is also important that these statistics can be attached to the newly created graph (representing another context). For instance, in simple random walk models the probability of going from one node to another node may be determined by the degrees of the nodes involved. These probabilities can be seen as attributes of the edges; ideally, a database query would be sufficient to put these probabilities in a graph. The closure property entails that we can also run queries on the attributes generated in this way. One such type of query could be a *probabilistic* query, which calculates new probabilities from probabilities present in the network.

Ranking Once an aggregate is computed, it can be desirable to rank results on aggregate values; for instance, one may not be interested in the centrality of all nodes, but only in the nodes that are most central. A database system should support such ranking queries and ideally be optimized to answer them more efficiently than by post-processing a sorted list of all results.

In the following two sections, we translate these requirements into a specification for a data representation and a data manipulation language.

4 Data representation

An important choice for any data management system is the representation of the data it operates on. For example, in Codd's relational database model [16], data is represented as sets of tuples. The challenge is to find a data model that is capable of storing any kind of information network, and that fulfils the requirements described in the previous section.

In its most basic form, an information network is a collection of objects with links between them. It is therefore natural to use objects and links as the basic building blocks for a network representation. However, as we have seen in the examples of the previous section, it is not always clear which concepts to consider as objects, and which as links. For example, is a publication an object, or a link between (co-)authors? Usually, the answer to this question depends on the application at hand. However, BiQL is intended as an application-independent data management system. This means that the data should be modelled in the most general way, and the term "object" should be taken as broad as possible.

Intuitively, we define it as *any entity that has meaning in reality*, or, less abstractly, as any entity that can have additional properties or roles assigned to it. Following this guideline, we only allow features on objects. That is, links are modelled as nothing more than ordered pairs of object identifiers, and they only express that two objects are connected.

Hence, the main choice that we have made is, in a sense, that also edges are represented as objects. An edge object is linked to the nodes it connects. Even though this may not seem intuitive, or could seem a bloated representation, the advantages of this choice outweigh the disadvantages because:

- by treating both edges and nodes as objects, we obtain simplicity and uniformity in dealing with attributes;
- it is straightforward to treat (hyper)edges as nodes (or vice versa);
- it is straightforward to link two edges, for instance, when one wishes to express a similarity relationship between two edges.

In this way, the data representation fulfills the requirements of simplicity, uniformity between nodes and edges, and flexibility.

However, not all objects in the network have the same meaning or role. In the bibliographic network, we had objects that represented authors, publications, citations, etc. In our data model, we use *domains* to indicate these categories of objects. Such a domain is a named set of objects. Objects can belong to any number of domains, for example, an ‘author’ in the bibliographic network can also be a ‘person’, or an ‘employee’, at the same time.

Apart from domain membership, each object can have an arbitrary set of features described by a list of name-value pairs.

5 Basic data manipulation

Now that we have a basic understanding of how the data is organized in the database, we can focus on manipulating this information. In this section, we give a general overview of BiQL’s query language. For a more in-depth discussion on the query language and its underlying operations we refer the reader to [19, chapter 6].

The primary goal of BiQL is to manipulate a network by querying, analyzing, and modifying its objects and links. The main operations offered by the query language are

- adding an existing object to a new domain,
- adding links and attributes to an existing object,
- creating new objects (with links and attributes) and adding them to a new domain.

Each of these tasks can be specified as an `CREATE/UPDATE` query of the following form.

```
CREATE/UPDATE "domain name" <"variables"> {"object properties"}
FROM "selection from domains"
WHERE "predicate on attributes of objects"
LIMIT "k" ON "sorting criteria"
```

For example, the query

```
UPDATE Pubs2010<p>
FROM Publ p
WHERE p.year = 2010
```

creates a new domain `Pubs2010` that contains all articles published in 2010. The `UPDATE` keyword indicates that existing objects are used instead of newly created ones. This means that all existing features for the objects are preserved (unless they are overwritten by an object property definition in the query).

In general, a query in BiQL consists of the following statements:

The `FROM` statement defines the structural component of the query and introduces variables that can be used in the other statements.
The `WHERE` statement defines constraints on these variables based on the features of the objects.
The `CREATE/UPDATE` statement describes the output of the query, that is, how objects should be created or updated based on the retrieved information, and where they should be stored.
The `LIMIT` statement allows for ranking the results of a query and returning only the top k results.

FROM statement The primary function of the `FROM` statement is to define a graph pattern that must be matched in the network. Within this pattern, variables are defined that can be used in the other parts of the query. In a sense, this statement has the same role as SQL's `FROM` statement, that is, determining which sources of information to use, and how these sources are related. In BiQL, the `FROM` statement consists of a list of path expressions, where each path expression consists of an alternating sequence of object definitions and link expressions indicating how the objects are connected. For example, a co-authorship relation in the publication network can be expressed as the following sequence of objects and links.

```
Author a -> AuthorOf -> Publ p <- AuthorOf <- Author b
```

Every object is described by a domain it belongs to (e.g. `Author`), and, optionally, a variable name (e.g. `a`). The arrows between the objects indicate the direction of the links between them. A path expression by itself can only express a sequence. However, the `FROM` statement can contain multiple path expressions that can be connected by references. For example, if we are interested in co-authorship within certain topics, we can include the domain 'Keyword' in the graph pattern by using the path expression

```
#p -> HasKeyword -> Keyword k
```

where `#p` is a reference to the variable p in the previous expression. This pattern is shown in Figure 4. Variable references can also be used to point to variables defined in the same path expression, for example, for expressing cycles.

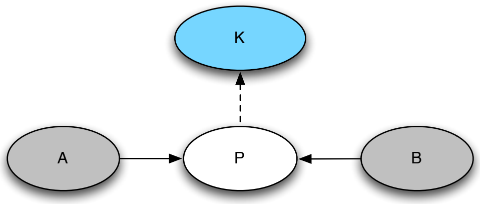


Fig. 4: Example of a graph pattern

Many problems in network analysis are based on finding paths of arbitrary length. To express such paths, we use regular expression operators. For example, the path expression

```
Node (-> Edge -> Node)* -> Edge -> Node
```

defines a path as an alternating sequence of nodes and edges of arbitrary length. To specify constraints on this path we can use *list variables*. These variables capture a sequence of objects instead of a single object. For example, we can use this to restrict the length of a path as shown in the following example.

```
FROM Node (-> Edge [e] -> Node)* -> Edge [e] -> Node
WHERE length(e) <= 4
```

The list aggregate `length` counts the number of objects assigned to the variable `e`.

`WHERE` statement The `FROM` statement generates a set of tuples corresponding to the possible assignments of objects in the network to the variables defined in the path expressions. The `WHERE` statement of the query can impose further constraints on this set of tuples based on the features of the resulting objects. For example, given the path expressions above, one can express the constraints

```
WHERE p.year = 2009 AND k.keyword = 'Data Mining'
```

to find only publications from 2009 in the field of data mining. This statement is equivalent to the `WHERE/HAVING` statements of SQL.

`CREATE/UPDATE` statement The previous operations produce a set of tuples. However, the final result of the query should fit into BiQL's data representation model. This means that this set of tuples should be transformed into a set of objects, links and domains. This transformation is defined in the `CREATE/UPDATE` statement, which is written as

```
CREATE/UPDATE DomainName<Var1,Var2,...> {<object properties>}.
```

A key part of this statement is the *partition operation* `<Var1,Var2,...>`, which splits the set of tuples and creates a separate partition for each distinct combination of the variables *Var1*, *Var2*,.... The final results of the query will contain a separate object for each of these partitions. The features and links of this object are described by the object properties. After construction, the set of objects is stored in the domain with the given name. This partition operation is comparable to the `GROUP BY` statement in SQL.

For example, if we want to define the co-author relationship, we can use the following query.

```
CREATE CoAuthor<a,b> { a ->, -> b, strength: count<p> }
FROM Author a -> AuthorOf -> Publ p <- AuthorOf <- Author b
WHERE a != b
```

This query creates a new object for each pair of authors a and b that have published at least one article together, that is, for whom the path expression can be mapped onto part of the network. The object properties specify that this new object is linked to both authors and that it contains an attribute `strength` indicating the number of articles the authors have co-written. The created objects are added to the new domain `CoAuthor`.

The partition operator is also used in the calculation of aggregate functions, for example, the function `count<p>` counts the number of distinct p, that is, the number of partitions `<p>` creates. Other aggregates include `sum<...>(expr)`, `min<...>(expr)`, `max<...>(expr)`, etc.

LIMIT statement Apart from feature-based selection, BiQL also supports rank-based selection through the `LIMIT` statement.

```
LIMIT k BY criteria
```

For example, we can select the three strongest co-authorship relationships using the statement

```
LIMIT 3 BY count<p> DESC
```

where `DESC` indicates a descending sort order. This statement is a *global* limit statement, that is, it is used to reduce the number of objects returned by the query. The operation of this statement is comparable to the `ORDER BY` statement in SQL, combined with a statement for selecting the top-k results (e.g. `FETCH FIRST` in SQL.2008 [25]).

In this section, we provided a limited overview of the features present in the BiQL query language. For an extensive description of this query language and its operational model, we refer the reader to [19, chapter 6].

6 Illustrative examples

In Section 2, we introduced a list of key tasks that we want to support in BiQL. We now revisit this list to evaluate BiQL's capabilities. Unless stated otherwise, each of these queries can be evaluated using the prototype implementation on the ILPnet2 publication database. This database is structurally similar to the network shown in Figure 1.

1. Introduce new relationships in the network. Throughout this chapter, we have repeatedly used the co-author relationship as an example of a new relationship. Here, we express this relationship as a connection between authors that have more than one publication in common.

```
CREATE CoAuthor<a,b> { a->, b<-, strength: count<p> }
FROM Author a -> AuthorOf -> Publ p <- AuthorOf <- Author b
WHERE count<p> > 1 AND a != b
```

Another example introduces the 'InArea' relation, which expresses whether an author has published a paper within a certain research area (indicated by a 'Keyword').

```
CREATE InArea<a,k> { a->, k<-, weight: count<pk>/count<p> }
FROM Author a -> AuthorOf -> Publ pk -> HasKeyword -> Keyword k,
     #a -> AuthorOf -> Publ p
```

The attribute 'weight' indicates the fraction of the author's publications that contain this keyword.

2. Find connections between objects. Using the 'InArea' and 'CoAuthor' relations, we can express how far an author is removed from any given research area.

```
CREATE RelatedToArea<a,k>{ a->, k<-, distance: min<b>(length(b))}
FROM Author a (-> CoAuthor -> Author [b])* -> InArea -> Keyword k
```

The expression `min<b>(length(b)` computes the length of the shortest path (i.e. the number of intermediate authors) from a specific author to a specific keyword.

3. Find the transitive closure of a relation. The previous query already used the transitive closure of the 'CoAuthor' relation to find a relationship between authors and research areas. We can also use such a relationship to determine the size of the neighborhood of an author.

```
UPDATE <a> { a->, b<-, networksize: count<b> }
FROM Author a (-> CoAuthor [co] -> Author)*
                                 -> CoAuthor [co] -> Author b
WHERE length(co) < 4
```

4. Rank results Often we are interested in finding the top-k results according to some criteria. For example, we might be interested in the top 3 authors with most co-authors.

```
SELECT[3] <a>
FROM Author a -> CoAuthor co
LIMIT 3 BY count<co> DESC
```

We can also find the authors with the largest network of co-authors up to a certain distance.

```
SELECT <a> { network_size: count<b> }
FROM Author a -> CoAuthor [co] ->
          (Author -> CoAuthor [co] ->)* -> Author b
WHERE length(co) < 4
LIMIT 3 BY count<b> DESC
```

5. Calculate network analysis metrics Another interesting task is calculating network analysis metrics such as centrality measures. Perhaps the simplest centrality measure is degree centrality which calculates, for a given node v, the fraction of all nodes that v is connected to. In BiQL, this measure can be calculated, for all authors simultaneously, using the following query.

```
UPDATE <a> { Cdegree: count<b>/(count<n> - 1)}
FROM Author a -- CoAuthor -- Author b, Author n
```

Another common centrality measure is closeness centrality, which involves determining the length of the shortest path to all other nodes in the network. First let us define the notion of shortest path between two authors using the co-authorship relation.

```
CREATE ShortestPath<a,b>{ a->, b<-, len: min<co>(length(co))}
FROM Author a -> CoAuthor [co] ->
          (Author -> CoAuthor [co] ->)* -> Author b
WHERE a != b
```

This query creates for each pair of (connected) authors a and b an object with as attribute the length of a shortest path between them. Using these new objects, we can easily calculate the closeness centrality as follows.

```
UPDATE <a>{ Cclose: 1/sum<b>(min[4]<sp>(sp.len))}
FROM Author a -> ShortestPath sp -> Author b
```

[3] In our prototype implementation, a `SELECT` query can be used to output a list of results without causing changes to the database.

[4] Given the definition of `ShortestPath` we expect the variable `sp` to be uniquely identified when `a` and `b` are fixed (i.e. there is only one shortest path length between two given nodes). However, BiQL currently does not support such type of constraint reasoning across queries. This is why we need the additional aggregation `min<sp>` even though there is only one value for `sp.len` in this context.

Another type of centrality measure is the betweenness centrality. This measure expresses the importance of a node based on its occurrence on the shortest paths in the network. In BiQL this measure can be expressed using the following two queries. The first query computes the length of the shortest path between each pair of authors and calculates how many paths of this length there are.[5]

```
CREATE ShortestPathCount<a,b> { a ->, b <-,
             count: count<co>, length: min<co>(length(co)) }
FROM Author a -> CoAuthor [co] ->
                    (Author -> CoAuthor [co] ->)* -> Author b
LIMIT 1 KEYS ON length(co) ASC
```

The second query uses this information to calculate the betweenness centrality of a node v as a fraction of shortest paths in the network that contain v.

```
UPDATE <v> { Cb: sum<s,t>((sv.count*vt.count)/st.count) }
FROM Author s -> ShortestPathCount sv -> Author v ->
                           ShortestPathCount vt -> Author t,
      #s -> ShortestPathCount st -> #t
WHERE st.length = sv.length + vt.length
  AND s != t AND s != v AND t != v
```

This query uses the calculation approach for betweenness centrality described in [3, section 3].

6. Introduce weights and probabilities Another important aspect of BiQL is its ability to deal with probabilistic networks. To illustrate this, we first need to introduce probabilities in our network. For this we assume that the information in the network is very unreliable by stating that for each publication in the network there is only 10% probability that it actually exists. Under this assumption we can attach a probability to each co-author connection using the following query.

```
UPDATE <co>{ prob: 1-(0.9^co.strength) }
FROM CoAuthor co
```

We can now calculate for each pair of authors the probability that they are connected using the probabilistic aggregate `problog_connect`.

```
CREATE ProbConnect<a,b>{a->, b<-, prob: problog_connect(co.prob)}
FROM Author a -> CoAuthor [co] ->
          (Author -> CoAuthor [co] ->)* -> Author b
WHERE a != b
```

The `problog_connect` aggregate uses ProbLog's [18] approach to calculate the connection probability between each pair of nodes in the network.

[5] For clarity, we omitted the extra aggregation operations on the variables `sv`, `vt` and `st` as described in the previous footnote.

7. Discover bisociations We can use this domain in combination with the shortest path to find authors that are very likely connected, but that are relatively far apart in the co-author network.

```
SELECT <a,b,pc,sp>{nameA: a.name, nameB: b.name,
                                prob: pc.prob, dist: sp.length }
FROM Author a -> ProbConnect pc -> Author b,
      #a -> ShortestPath sp -> #b
WHERE sp.length > 2
LIMIT 3 BY pc.prob DESC
```

Another example of bisociative discovery consists of finding bridging nodes between different domains. In Example 5 we described betweenness centrality. If we modify the second part of that query we can express the interdomain betweenness centrality as the occurrence of a node on the shortest paths between *concepts in different domains*.

```
UPDATE <v> { Cb: sum<s,t>((sv.count*vt.count)/st.count) }
FROM DomainA s -> ShortestPathCount sv -> DomainC v ->
                            ShortestPathCount vt -> DomainB t,
      #s -> ShortestPathCount st -> #t
WHERE st.length = sv.length + vt.length
  AND s != t AND s != v AND t != v
```

8. Apply external algorithms on the network In the final task of section 2, we want to apply external algorithms on the networks in BiQL. Unfortunately, there are still many open questions on how this integration should work in practice. However, instead of providing integration of external tools within BiQL, we have integrated BiQL in the data analysis integration platform KNIME [4]. Through this platform, networks can be passed from BiQL to external algorithms and back, allowing BiQL to be used as part of a broader knowledge discovery process.

7 Related work

7.1 Knowledge Discovery

Graph mining Graph mining aims at extending the field of pattern mining towards graphs. Most graph mining techniques work in the transactional setting, that is, on data consisting of sets of graphs. As in item set mining, the focus lies on finding subgraphs that, for example, occur frequently in such a set [33, 41, 24]. However, many other interestingness measures have been translated towards graph patterns (e.g. correlated patterns [8, 15]), and new graph-specific measures have been introduced (e.g. for finding quasi-cliques [42]). Several techniques have been developed that target subsets of graph representations, such as sequences or trees [39]. Recently, there has been increasing interest in applying graph mining techniques to the network setting, that is, to a single graph [10, 9, 26].

Network analysis Network analysis is concerned with analyzing the properties of networks, by use of graph theoretical concepts such as node degrees and paths [6]. The primary tool in network analysis are measures such as centrality [36], and specialized algorithms for calculating them efficiently have been developed (e.g. [5]). This field has also gained a lot of interest in domains outside computer science, for example, in social sciences (social network analysis) [40].

Another part of network analysis focusses on the spread of information in a network. This can be used to determine the importance of, for example, web pages on the World Wide Web [7], or to analyze the transmission of infectious diseases [38].

7.2 Databases

General-purpose database systems The best-known general purpose database systems are based on Codd's relational data model [16]. Many of these database systems (e.g. Oracle Database, Microsoft SQL Server, MySQL, PostgresQL) use (a variant of) ISO SQL [25] as the query language of choice. Datalog [13] is an alternative query language that is based on first order logic. Syntactically, it is a subset of Prolog restricted as to make efficient query answering possible.

A more recent development is that of object-oriented database systems and query languages such as OQL (Object Query Language) [12]. These systems use objects instead of tuples, and they allow for nested objects. Part of the OQL standard focusses on a tight integration with object-oriented languages such as Java and C++. However, due to the overall complexity of object databases, there are few systems that fully support the OQL standard.

Recently, there is a increasing interest in so-called NoSQL databases. These database systems focus on applications that require extremely large databases. Such databases typically use non-relational representations specialized for specific applications, such as Google's BigTable [14] or Facebook's Cassandra [11]. Current graph databases such as Dex [31] and Neo4J [32] also fall under this category, and arguably BiQL does as well.

Graph query languages A number of query languages for graph databases have been proposed, many of which have been described in a recent survey [2]. However, none of these languages was designed for supporting the knowledge discovery process and each language satisfies at most a few of the requirements mentioned in Section 3. For instance, GraphDB [20] and GOQL [37] are based on an object-oriented approach, with provisions for specific types of objects for use in networks such as nodes, edges and paths. This corresponds to a more structured data model that does not uniformly represent nodes and edges. In addition, these languages target other applications: GraphDB has a strong focus on representing spatially embedded networks such as highway systems or power lines, while GOQL [37], which extends the Object Query Language (OQL), is meant for querying and traversing paths in small multimedia presentation graphs. Both languages devote a lot of attention to querying and manipulating paths: for example, GraphDB supports regular expressions and path rewriting operations.

GraphQL [22] provides a query language that is based on formal languages for strings. It provides an easy, yet powerful way of specifying graph patterns based on graph structure and node and edge attributes. In this model graphs are the basic unit and graph specific optimizations for graph structure queries are proposed. The main objective of this language is to be general and to work well on both large sets of small graphs as well as small sets of large graphs. However, extending existing graphs is not possible in this language; flexible contexts are not supported.

PQL [27] is an SQL-based query language focussed on dealing with querying biological pathway data. It is mainly focussed on finding paths in these graphs and it provides a special path expression syntax to this end. The expressivity of this language is, however, limited and it has no support for complex graph operations.

GOOD [21] was one of the first systems that used graphs as its underlying representation. Its main focus was on the development of a database system that could be used in a graphical interface. To this end it defines a graphical transformation language, which provides limited support for graph pattern queries. This system forms the basis of a large group of other graph-oriented object data models such as Gram [1] and GDM [23].

Hypernode [29] uses a representation based on hypernodes, which make it possible to embed graphs as nodes in other graphs. This recursive nature makes them very well suited for representing arbitrarily complex objects, for example as underlying structure of an object database. However, the data model is significantly different from a traditional network structure, which makes it less suitable for modeling information networks as encountered in data mining.

A similar, but slightly less powerful representation based on hypergraphs is used in GROOVY [30]. This system is primarily intended as an object-oriented data model using hypergraphs as its formal model. It has no support for graph specific queries and operations.

More recently, approaches based on XML and RDF are being developed, such as SPARQL [34]. They use a semi-structured data model to query graph networks in heterogenous web environments; support for creating new nodes and flexible contexts is not provided.

While most of the systems discussed here use a graph-based data model and are capable of representing complex forms of information, none of them uses a uniform representation of edges and nodes (and its resulting flexible contexts), nor supports advanced aggregates.

Graph databases Whereas the previous studies propose declarative query languages, recently several storage systems have been proposed that do not provide a declarative query language. Notable examples here are Neo4J [32] and DEX [31], which provide Java interfaces to graphs persistently stored on disk. For Neo4J an alternative programming language called Gremlin is under development [35].

Graph libraries Finally, in some communities, Java or C++ libraries are used for manipulating graphs in the memory of the computer (as opposed to the above graph databases which support typical database concepts such as transactions). Examples are SNAP [28] and igraph [17].

8 Conclusions

In this article, we gave an introduction to BiQL, a novel system for representing, querying and analyzing information networks. The key features of this system are:

– It uses a *simple, yet powerful representation model*. Using only objects (with attributes), links (as pairs of objects), and domains (as named sets of objects), it is capable of representing a wide variety of network types, such as labelled graphs, directed hypergraphs, and even sets of graphs.
– Its query language is *declarative*. This means that the queries only describe what the results should be, but not how they should be obtained. This makes the language more accessible to the average user.
– Its query language uses a powerful mechanism for *expressing graph patterns based on regular expressions*. This makes it possible to, for example, express paths of arbitrary length.
– Its query language allows for the use of *nested aggregates* with a syntax that closely resembles mathematical notation. These aggregates allow the user to perform all kinds of analysis tasks, such as calculating distances and centrality measures.
– Its query language provides a powerful mechanism for *object creation*, which makes it possible to return structured output from a query. However, the result of a query always produces a new network that can be queried again.
– The system itself is developed *from a knowledge discovery perspective*. It focusses on providing specific support for knowledge discovery operations such as network analysis, ranking, and tool integration.

In this chapter, we focussed on defining a data model and the syntax and semantics of the corresponding query language. In future work, the main challenge is to develop a query optimization model that would form the basis of a scalable implementation of the BiQL system.

References

1. Bernd Amann and Michel Scholl. Gram: a graph data model and query language. In *Proceedings of the ACM conference on Hypertext*, pages 201–211. ACM, 1993.
2. Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Computing Surveys*, 40(1):1–39, 2008.
3. Vladimir Batagelj. Semirings for social network analysis. *Journal of Mathematical Sociology*, 19(1):53–68, 1994.

4. M.R. Berthold, N. Cebron, F. Dill, T.R. Gabriel, T. Kötter, T. Meinl, P. Ohl, C. Sieb, K. Thiel, and B. Wiswedel. Knime: The Konstanz information miner. *Data Analysis, Machine Learning and Applications*, pages 319–326, 2008.
5. U. Brandes. A faster algorithm for betweenness centrality. *The Journal of Mathematical Sociology*, 25(2):163–177, 2001.
6. Ulrik Brandes and Thomas Erlebach, editors. *Network Analysis*, volume 3418 of *Lecture Notes in Computer Science*. Springer, 2005.
7. Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.
8. Björn Bringmann. *Mining Patterns in Structured Data*. PhD thesis, Katholieke Universiteit Leuven, 2009.
9. Björn Bringmann and Siegfried Nijssen. What is frequent in a single graph. In Takashi Washio, Einoshin Suzuki, Kai Ting, and Akihiro Inokuchi, editors, *Proceedings of Advances in Knowledge Discovery and Data Mining, 12th Pacific-Asia Conference (PAKDD 2008)*, volume 5012 of *Lecture Notes in Computer Science*. Springer Berlin-Heidelberg, 2008.
10. T. Calders, J. Ramon, and D. Van Dyck. Anti-monotonic overlap-graph support measures. In *Proceedings of the 8th International Conference on Data Mining*, pages 73–82. IEEE, 2009.
11. Cassandra. The Apache Cassandra project. http://cassandra.apache.org.
12. Roderic G.G Cattell and Douglas K. Barry, editors. *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann Publishers, 2000.
13. S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about DataLog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, 1(1):146–166, 1989.
14. Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. In *Seventh Symposium on Operating System Design and Implementation*, 2006.
15. H. Cheng, X. Yan, Jiawei Han, and Hsu C.-W. Discriminative frequent pattern analysis for effective classification. In *Proceedings of the 23rd International Conference on Data Engineering*, pages 716–725. IEEE, 2007.
16. Edgar F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.
17. Gábor Csárdi and Tamás Nepusz. The igraph library. http://igraph.sourceforge.net/.
18. Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. ProbLog: A probabilistic prolog and its application in link discovery. In Manuela M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2462–2467, 2007.
19. Anton Dries. *Data streams and information networks: a knowledge discovery perspective*. PhD thesis, Katholieke Universiteit Leuven, 2010.
20. Ralf Hartmut Güting. GraphDB: Modeling and querying graphs in databases. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 297–308. Morgan Kaufmann, 1994.
21. Marc Gyssens, Jan Paredaens, Jan Van den Bussche, and Dirk van Gucht. A graph-oriented object database model. *IEEE Transactions on Knowledge and Data Engineering*, 6(4):572–586, 1994.

22. Huahai He and Ambuj K. Singh. Graphs-at-a-time: query language and access methods for graph databases. In Jason Tsong-Li Wang, editor, *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 405–418. ACM, 2008.
23. Jan Hidders. Typing graph manipulation operations. In *Proceedings of the 9th International Conference on Database Theory*, volume 2572 of *Lecture Notes in Computer Science*, pages 394–409. Springer-Verlag, 2003.
24. Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. Complete mining of frequent patterns from graphs: Mining graph data. *Machine Learning*, 50(3):321–354, 2003.
25. International Organization for Standardization. SQL Language. ISO/IEC 9075(1-4,9-11,13,14):2008, 2008.
26. Michihiro Kuramochi and George Karypis. Finding frequent patterns in a large sparse graph. *Data Mining and Knowledge Discovery*, 11(3):243–271, 2005.
27. Ulf Leser. A query language for biological networks. *Bioinformatics*, 21(2):33–39, 2005.
28. Jure Leskovec. The SNAP library. http://snap.stanford.edu/snap/.
29. Mark Levene and Alexandra Poulovassilis. The hypernode model and its associated query language. In *Proceedings of the fifth Jerusalem conference on Information Technology*, pages 520–530. IEEE Computer Society Press, 1990.
30. Mark Levene and Alexandra Poulovassilis. An object-oriented data model formalised through hypergraphs. *Data and Knowledge Engineering*, 6(3):205–224, 1991.
31. N. Martínez-Bazan, V. Muntés-Mulero, S. Gómez-Villamor, J. Nin, M. Sánchez-Martínez, and J. Larriba-Pey. Dex: high-performance exploration on large graphs for information retrieval. In Mário J. Silva, Alberto H. F. Laender, Ricardo A. Baeza-Yates, Deborah L. McGuinness, Bjørn Olstad, Øystein Haug Olsen, and André O. Falcão, editors, *Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management*, pages 573–582. ACM, 2007.
32. Neo Technology. The Neo4J project. http://neo4j.org.
33. Siegfried Nijssen. *Mining Structured Data*. PhD thesis, Universiteit Leiden, 2006.
34. Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF. http://www.w3.org/TR/rdf-sparql-query/, 2008.
35. Marko A. Rodriguez. Gremlin. http://wiki.github.com/tinkerpop/gremlin/.
36. Gert Sabidussi. The centrality index of a graph. *Psychometrika*, 31:581–603, 1966.
37. Lei Sheng, Z.Meral Ozsoyoglu, and Gultekin Ozsoyogly. A graph query language and its query processing. In *Proceedings of the 15th International Conference on Data Engineering*, pages 572–581. IEEE Computer Society, 1999.
38. Sven Van Segbroeck, Francisco C. Santos, and Jorge M. Pacheco. Adaptive contact networks change effective disease infectiousness and dynamics. *PLoS Computational Biology*, 6(8):1–10, 2010.
39. Takeshi Washio, Joost N. Kok, and Luc De Raedt, editors. *Advances in Mining Graphs, Trees and Sequences*, volume 124 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2005.
40. Stanley Wasserman and Katherine Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.
41. Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE Conference on Data Mining*, page 721. IEEE Computer Society, 2002.

42. Zhiping Zeng, Jianyong Wang, Lizhu Zhou, and George Karypis. Coherent closed quasi-clique discovery from large dense graph databases. In Tina Eliassi-Rad, Lyle H. Ungar, Mark Craven, and Dimitrios Gunopulos, editors, *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 797–802. ACM, 2006.