

Information Discovery in Loosely Integrated Data

Heasoo Hwang*, Andrey Balmin, Hamid Pirahesh, Berthold Reinwald

IBM Almaden Research Center

650 Harry Rd

San Jose, CA 95120

heasoo@cs.ucsd.edu, {abalmin, pirahesh, reinwald}@us.ibm.com

ABSTRACT

We model heterogeneous data sources with cross references, such as those crawled on the (enterprise) web, as a labeled graph with data objects as typed nodes and references or links as edges. Given the labeled data graph, we introduce flexible and efficient querying capabilities that go beyond existing capabilities by additionally discovering meaningful relationships between objects that satisfy keyword and/or structured query filters. We introduce the *relationship search* operator that exploits the link structure between data objects to rank objects related to the result of a filter. We implement the relationship search operator using the ObjectRank [1] algorithm that uses the random surfer model. We study several alternatives for constructing summary graphs for query results that consist of individual and aggregate nodes that are somehow linked to qualifying result nodes. Some of the summary graphs are useful for presenting query results to the user, while others could be used to evaluate subsequent queries efficiently without considering all the nodes and links in the original data graph.

ACM CLASSIFICATION

H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval - Query formulation; Information filtering; Search process; Selection process; H.2.4 [Database Management]: Systems - Query processing; Textual databases

1. INTRODUCTION

Integration of data (structured, semi-structured, and unstructured) continues to be a hot topic in the research community and industry. Recent efforts on this topic focus on automatic or semi-automatic discovery of common entities and relationships across disparate data collected by crawling thousands of data sources on the (enterprise) web. The heterogeneous nature of real world data and the multitude of ways of situational integration of data call for extensible data structures and creative ways for data retrieval across disparate data sources. By analogy, the IR community addressed this issue by introducing search engines that index crawled web documents. An example of a query is as follows. A product manager looking for employees in a certain department who somehow (directly or indirectly) contributed to a shipped

product, may want to take the product plan data coming from a content repository and dynamically combine it with the company employee data to find employees. The product manager expects to find employees who, for example, owned components of the product, developed components, or consulted employees on the development of components. Similar to users of search engines, these users are typically after data retrieval with "high recall" rather than "high precision". Since large amounts of data may be related to the query, it is important to be able to summarize the results by identifying the highest ranking objects and relationships individually, and aggregating the less important ones.

In this demonstration, we provide flexible and efficient querying capabilities for highly heterogeneous data sources that reference each other. Our data sources are represented as a labeled graph with data objects as typed nodes and references or relationships as edges. We present a graph search system that evaluates queries with keywords, structured predicate filters, and a novel *relationship search* operator, $\sim$, that exploits the references between data objects to rank objects related to the result of a filter. The ranking is implemented by the ObjectRank [1] algorithm that uses the random surfer model.

The query result is presented as a graph with qualifying data nodes and additional nodes necessary to interconnect qualifying nodes. As a query result graph might potentially get very large, we introduce the notion of summary graphs, that consist of individual and aggregate nodes of the data graph that are related to the objects in the query results. We demonstrate a number of algorithms for constructing summary graphs for query results. Some of these summary graphs are useful for presenting query results to the user, while others could be used to evaluate subsequent queries efficiently without re-considering all the nodes and edges in the original data graph. We observe that executing a subsequent query on a summary graph of the first query is not only faster than executing both queries on the full data graph, but also usually produces a better qualitative result.

2. SCENARIO

In our demonstration, we assume two independent databases. A product design database that contains the description of product features (also known as *lineitems*), and an employee database with the company organigram, work locations, etc. We assume an elaborate data integration phase that analyses the databases [2], performs entity resolution [3], and identifies data objects and relationships [4] (Figure 1). We consider three types of objects: People, LineItems, and Comments, and four types of relationships: a LineItem is owned by a Person, a Comment belongs to a LineItem, and a comment is written and answered by a Person. We process queries of the following nature:

* Work performed at IBM Almaden Research Center. Current Address: Department of Computer Science and Engineering, UC San Diego

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGMOD '07, June 12–14, 2007, Beijing, China.

Copyright 2007 ACM 978-1-59593-686-8/07/0006...\$5.00.

Example 1: Find the features of DB2 impacted by people who work at the Almaden.

To express such queries, we define a query language that combines filter terms with a relationship search operator, $\sim$, that traverses the edges of the instance graph.

We use three types of filters to subset the data objects: keyword, object type, and XPath expression. A filter term returns a set of data object IDs that satisfy the condition. For example, `<lineitem>` is a filter term that selects all lineitem objects, and `/person/location[text()='Almaden']` is a filter term that finds all the person entries with location “Almaden”.

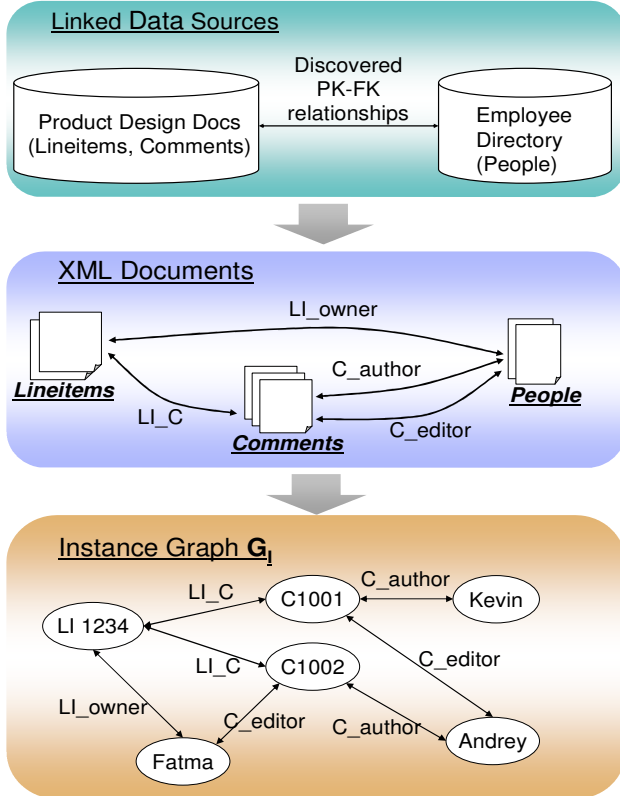


Figure 1: Constructing the Data Graph.

We define a relationship search operator, $\sim$, that uses the edges between data objects in the instance graph. Given a subset of nodes of the graph (the result of a filter term), a relationship search operator calculates the rank of each node in the graph with respect to the subset. A relationship search term is composed of a relationship search operator followed by the filter term that selects its input subset of data objects. For example, `$\sim$ /person/location[text()='Almaden']` is a relationship search term that ranks all the objects in the dataset based on the strength of their relationships to Almaden people.

3. ARCHITECTURE

Our system is implemented on top of a commercial DBMS (IBM DB2® V9) which supports native persistence of XML documents

and querying XML documents using XPath and keyword containment predicates. The architecture of our demonstration is shown in Figure 2.

We process the query of Example 1 by first querying the XML documents to find all Person objects that have work location “Almaden”. We then use the qualifying objects as a *base set*, i.e., as starting points of random walks on the instance graph, and rank all the objects in the graph based on the probability that a random walk visits the object at a given moment. The random walks computation is implemented using the iterative ObjectRank algorithm.

The system computes relationship searches not only on the original instance graph, but also on *summary graphs* produced from results of prior queries. A summary graph of a query consists of all the objects that have a score above a certain threshold and possibly aggregate objects. An aggregate object is constructed from a set of objects of the same type that share relationships to objects already in the summary graph. All individual objects that form an aggregate object have scores below the threshold, but if the sum of their scores surpasses the threshold, the aggregate qualifies for the result. We observe that for related search terms, characterized by high overlap of base sets, running the relationship search on the summary graphs, instead of the full instance graph, improves both the system performance and the quality of the result.

In the last step of the query processing, we identify the top-K highest ranked objects and generate a special kind of summary graph that we describe next.

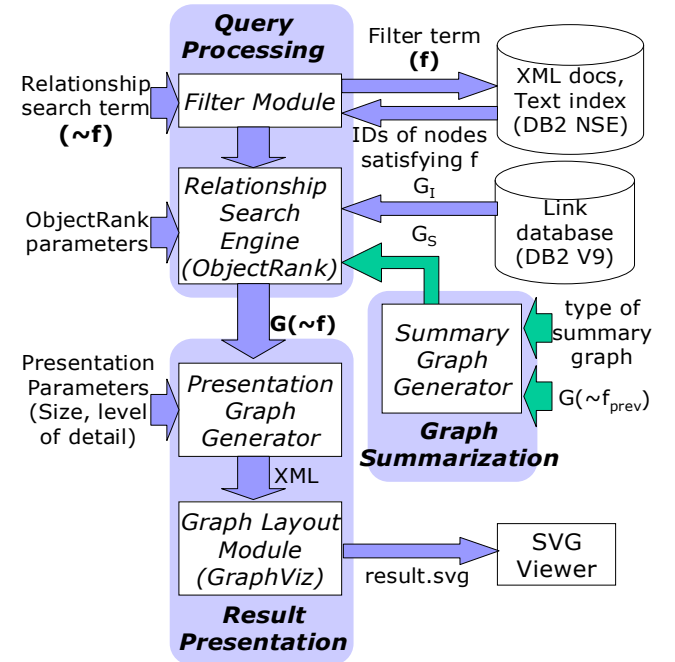


Figure 2: Demo Architecture

4. PRESENTING THE RESULTS

Our system presents results as graphs that are designed to serve two purposes: (1) provide some context to result objects and (2) show the strength of references from other objects.

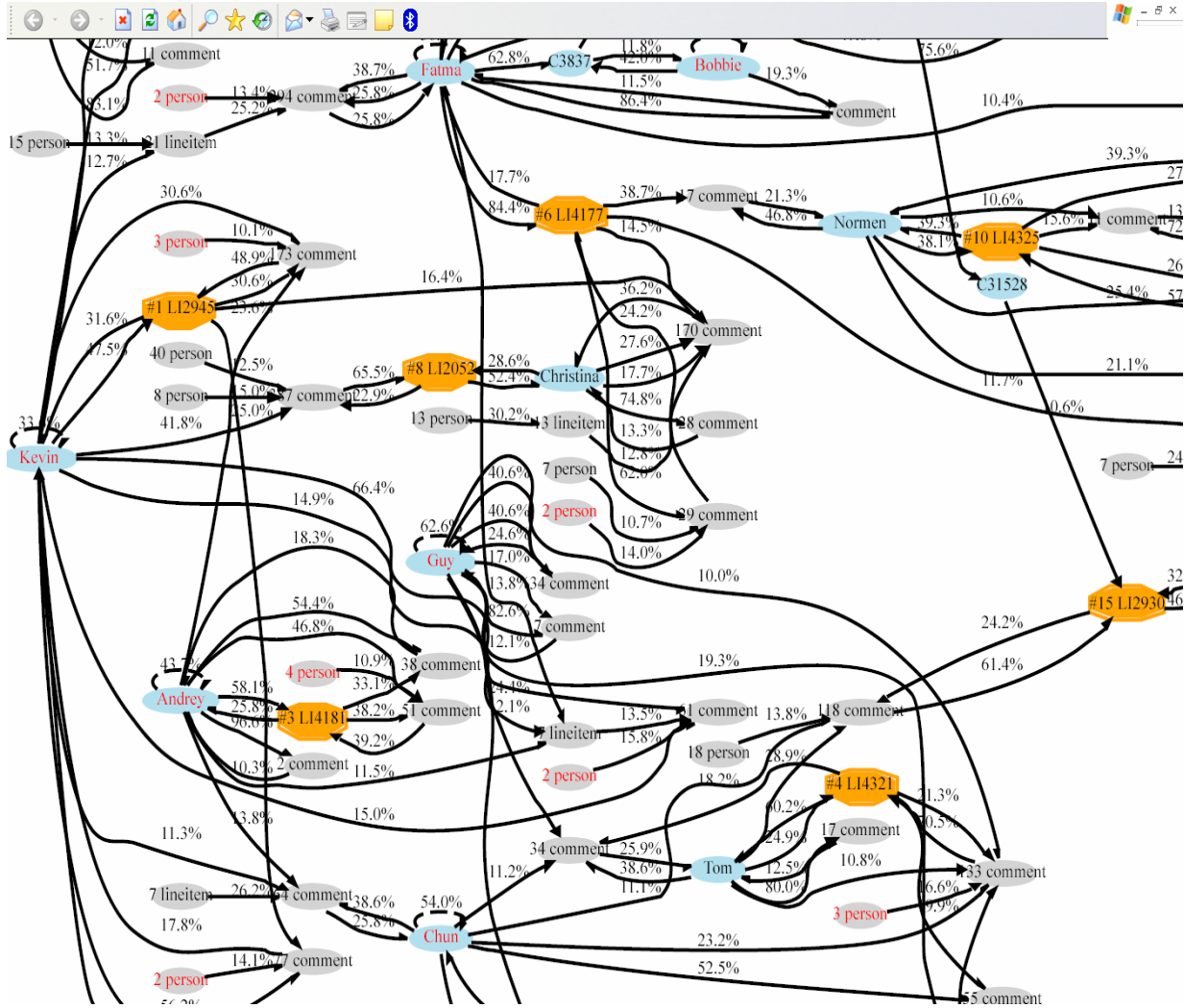


Figure 3: Presentation of results with their context and justification.

Figure 3 shows part of the result presentation graph for the query in Example 1. The result objects of the query are the orange nodes in the graph and are labeled with “#X”. “X” indicates the rank of the object. E.g. ‘#1 LI2945’ is the top result for this query. We also show blue nodes for important people, comments, and lineitems that tightly relate to the result objects. Finally, we include aggregated nodes into the result presentation graph if we need to display important nodes that are transitively reference by many less important nodes. For example, we discover the strong connection between ‘Kevin’ and lineitem ‘#8 LI2052’ by constructing the aggregate node ‘387 comments’ that aggregates 387 comments answered by Kevin on the lineitem. The label of a node that passes the filter term is colored red. E.g. the label ‘Kevin’ is red because the node passes the ‘Almaden’ filter predicate. The weight on the edge from ‘Kevin’ to ‘#1 LI2945’ indicates that 47.5% of the score of ‘LI2945’ is due to the relationship with ‘Kevin’. The weight of the self edge of ‘Kevin’ node corresponds to the impact of the fact that the node satisfies the filter term on its score.

Presentation graphs are well suited for providing the “big picture” view of the query result by immediately showing clusters of

closely-connected people and lineitems. They also show important objects that were neither in the base set nor in the final result, but that are highly scored by the ObjectRank algorithm. Thus, the presentation graph essentially pre-fetches objects that are likely to be navigated by a user while further exploring the query result.

5. REFERENCES

- [1] Andrey Balmin, Vagelis Hristidis, Yannis Papakonstantinou: *ObjectRank: Authority-Based Keyword Search in Databases*. VLDB 2004: 564-575
- [2] Paul Brown, Peter J. Haas, Jussi Myllymaki, Hamid Pirahesh, Berthold Reinwald, Yannis Sismanis: *Toward Automated Large-Scale Information Integration and Discovery*. Data Management in a Connected World 2005: 161-180
- [3] IBM Entity Analytic Solutions (EAS) - Solution Overview www.ibm.com/software/data/db2/eas/
- [4] Yannis Sismanis, Paul Brown, Peter J. Haas, Berthold Reinwal: *GORDIAN: Efficient and Scalable Discovery of Composite Keys*. VLDB 2006: 691-7