Symbiosis in the Intranet: How Document Retrieval Benefits from Database Information

Christoph Mangold Holger Schwarz Bernhard Mitschang

Universität Stuttgart, IPVS
Universitätsstr. 38, D - 70569 Stuttgart
firstname.lastname@ipvs.uni-stuttgart.de

Abstract

The enterprise information space is split in two hemispheres. Documents contain unstructured or semistructured information; structured information is stored in databases. As regards the content, both kinds of information are complementary parts. However, enterprise information systems usually focus on one part, only. Our approach improves document retrieval in the intranet by exploiting the enterprise's databases. In particular, we exploit database information to describe the context of documents and exploit this context to enhance common full text search. In this paper, we show how to model and compute document context and present results on runtime performance.

1 Introduction

The enterprise information space is split in two disconnected hemispheres. The document side contains, e.g., reports, plans, meeting minutes, email notifications, and web pages. These documents are created and maintained according to enterprise workflows. In contrast, the database side comprises information for planning, operational management, controlling, etc. Although the information from both sides is complementary, it is not integrated on the system level. In the enterprise this results in a situation where the interconnection between both worlds has to be established in the heads of the employees, which is not only costly but also time-consuming.

In this paper, we connect documents with databases for the benefit of document retrieval in the intranet. Standard text search engines retrieve documents based on *content*. Our approach enables the search engine to additionally exploit document *context* that is retrieved from the enterprise's databases.

To represent documents and context we propose a graph-based data model. The enterprise's database schemas induce the structure of the graph. Graph nodes represent documents, relational tuples, and values. Edges represent attribute and foreign-key relationships. Since the graph models the context of documents we call it ContextGraph. From the ContextGraph, we derive the document contexts with a specialized shortest path algorithm. Then, we exploit the context information to extend standard full text search.

Figure 1 gives an impression of a ContextGraph. Since the motivation for this work originated from a manufacturing scenario we use an example from this area. The ContextGraph contains the document Doc11 that is a work instruction. In the enterprise, work instructions are listed in a document management system (DMS) with their URL. Furthermore the DMS stores the technician who maintains the work instruction, which is Bob Blue for Doc11. Doc11 itself can be found in the file system (FS). From the enterprise resource planning system (ERPS) we get that Bob Blue works in Team18 together with Ricky Red.

Consider the following simple scenario: For a new product, some of the work instructions that are maintained by Ricky Red's team need to be adapted. In particular, the release of the con-rod has to be delayed. Hence, the responsible employee feeds the query "release con-rod" AND "Ricky Red" to the search engine. However, since Ricky Red is the team leader, he does not maintain a single working instruction. In this case, a standard text-based search engine yields results that contain *either* "release con-rod" *or* "Ricky Red". This behavior is of no use for the employee since the result set is potentially large. He might skim through the result set to filter relevant documents manually or alternatively investigate possible query modifications. The latter may involve expert knowledge from co-workers or information from databases.

Our system helps the employee to directly access the desired information. It is based on the notion that in this situation it is desirable to retrieve, e.g. Doc11 that is maintained by a co-worker of Ricky Red and rank it as highly relevant. To achieve this, our approach uses

International Conference on Management of Data
COMAD 2006, Delhi, India, December 14–16, 2006

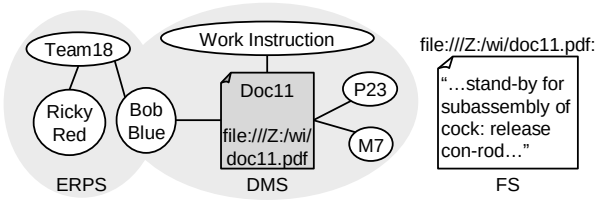

Figure 1: Small fraction of a ContextGraph.

information from the enterprise's ERPS.

The rest of the paper is organized as follows. In Sec. 2 we discuss related work, the ContextGraph we present in Sec. 3. In Sec. 4 we discuss the problem of determining the context. Sec. 5 addresses some performance issues and Sec. 6 concludes the paper.

2 Related Work

In the database area, related work focuses on database exploration. The typical scenario comprises a user who is running keyword queries against a database with unknown schema.

Goldman et al. propose the exploration of a Lorel/OEM database [9]. There, the user needs to specify two sets of objects: *find* and *near* objects. The system retrieves objects from the *find*-set according to their distance from objects in the *near*-set. In the relational database area, related approaches for keyword search support the user to find relationships in the database [1,12,13]. For a given set of search keywords, the systems return a set of joined tupels that contain the keyword, each. The BANKS system [3] is closely related to our approach since it represents a relational database as a graph where tuples and foreign key relationships are nodes and edges, respectively. For a given keyword query, the system returns a set of subgraphs each of which contains at least one hit node for each search keyword. The ranking of subgraphs is based on pre-computed node importances and edge weights.

All of the above-mentionend approaches have in common that they are limited to databases. None of them explicitly considers documents. In a scenario, where all documents are stored inside a database, they could be applied to solve the problem we described in Sec. 1. However, this scenario obviously can not be assumed for the majority of enterprises. Furthermore, the approaches are limited to treat document text just the same as relational data. I.e., document search would be an extension to database search. In contrast, we use the database context as an extension to full text which has important consequences on our system architecture. E.g., the system can retrieve documents with and without context at the same time.

There are ideas to integrate search engines with relational databases [7,10]. In contrast to our approach, they consider the results of the text search engine as a (virtual) table in the database system. The user interacts with the system by means of SQL queries. I.e., where our approach intends to exploit database information for information retrieval they aim at the opposite: To employ search engines as data sources for relational database systems. A somewhat similar approach enriches the result of SQL queries with documents retrieved by text search [18].

Recently, there has been considerable research in the area of XML retrieval systems, e.g. [2, 8, 11, 20]. None of them considers information from relational database systems. In general, they aim at exploiting both, the content and the logical structure of XML documents. In particular, they focus on the exploitation of the internal, hierarchical structure of XML documents. A query result always consists of document components. In contrast, our scenario is targeted towards external document context from databases. We do not presume hierarchical structures and, as opposed to path queries which yield document components, we aim at keyword search which results in complete documents.

In the scope of Semantic Web activities, there is considerable research on semantic search. Many approaches base on ontologies that need to be created by domain experts; some even require the user to be aware of ontology structures, e.g. [6]. Our approach differs from them in that domain knowledge is required only if the edge weights in the ContextGraph are not determined automatically. Many approaches base on handcrafted ontologies, e.g. [4, 19]. They are not applicable to our scenario since they mostly rely on concept hierarchies and linguistic information like synonyms. Furthermore, they do not cope with text fragments like document title or abstract but require ontological concepts that contain one or a few terms, only. Consequently, they are not capable to exploit the high quality information from the enterprise's databases. For these approaches, ontologies need to be created and custom-tailored for each application domain. This is not only costly but also provokes correctness and consistency issues. In contrast, our ContextGraph approach represents established and reliable information that is consistent with and relevant for the enterprise's business.

3 Modeling Document Context

In this section, we introduce the ContextGraph, a directed weighted graph that models the semantic distance between data items in an enterprise.

The ContextGraph (CG) is derived from the enterprise's databases (DB) by the following simple mapping: Each tuple in the DB becomes a node in the CG. Each attribute value belonging to the tuple also becomes a node that is connected to the tuple node. Tuple nodes are mutually interconnected according to foreign-key relationships in the DB. M:N-relationships

in the DB are denormalized and represented as graph edges. Documents are represented in CG by special nodes that contain the document's URL, only. Edges are weighted with a measure for semantic distance, which is out of the scope of this paper.

Rocha [17] proposed to model all attributes of a tuple as one single graph node. However, we favor the smaller node granularity, as it has been proposed, e.g., in the OEM data model [9] for two reasons: First, it permits to assign different semantic distances (see below) to attributes. And secondly, a graph that contains atomic nodes only is more flexible regarding the integration of non-relational sources (for a discussion of this issue refer to [9]).

Fig. 2 shows a portion of the DMS table WorkInstruction and portions of the ERPS tables Employee, and Team, that map to the ContextGraph in Fig. 1.

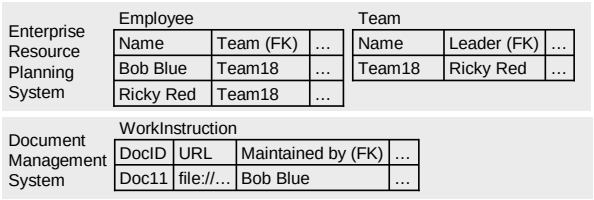

Figure 2: Example. Tables that map to the ContextGraph in Fig. 1, where FK denotes a foreign key.

To exploit the information represented in the ContextGraph, each document needs to be aware of its context. For two nodes $n_1, n_2 \in$ CG we define the *semantic distance* $\text{dist}(n_1, n_2)$ to be the weight of the shortest path from n_1 to n_2. Furthermore, let *cRange* be a positive value that denotes the context range. Then we define the *context* of a node $n \in$ CG as the set of nodes that are reachable from n within distance *cRange*:

$$\text{Context}(n) = \{v \in \text{CG} \mid \text{dist}(n, v) \leqq \mathit{cRange}\}.$$

Likewise, the context of a document d is the context of the node n_d that represents d.

4 Computing the Context

The problem of determining document context based on the ContextGraph is equivalent to find shortest paths in the graph. Clearly, an all-pairs-shortest-path (APSP) algorithm such as the Floyd-Warshall or the Johnson algorithm [5, p. 558] would yield the context information for each document node. From a theoretical viewpoint the complexity is $O(|V|^3)$ for Floyd-Warshall and $O(|V|^2 \cdot \lg |V| + |V| \cdot |E|)$ for Johnson. Another standard approach for APSP is to run $|V|$ single-source-shortest-path (SSSP) algorithms such as Dijkstra's algorithm which amounts to $O(|V| \cdot |E| \lg |V|)$ in a standard implementation.

Since the ContextGraph can grow arbitrarily large, we need a solution that is optimized towards our scenario. Based on the following two observations we find that our problem requires only a subset of the results of an APSP: First, we only need connectivity information for a subset of all nodes, i.e., the document nodes, only. Secondly, nodes that are far away from a document node have little relevance for the document. Hence, we only need the distances between a document node and the nodes in its context.

The execution of a restricted Dijkstra algorithm [5, p.527] on each document node exploits both optimizations. The Dijkstra algorithm is a greedy SSSP algorithm that discovers paths in the order of increasing weights. The restriction stops the algorithm when path weights exceed the cRange limit, i.e., when the context is identified.

We additionally considered the Hidden Path algorithm [14] that does not benefit from the first optimization but computes the context of all nodes. It works as $|V|$ individual Dijkstra algorithms in parallel, i.e., one for each graph node. Its running time benefits from the fact that information can be reused among the parallel Dijkstra algorithms. The Hidden Path algorithm also discovers paths in increasing weight order. We designed a restricted variant of the original Hidden Path algorithm which profits from the second optimization. In the next Section, we show some results of our experiments with the restricted HiddenPath algorithm and compare it with Dijkstra's algorithm.

5 Evaluation

In this section, we present some results of our runtime experiments. Any discussion about the architecture or implementation of our system is out of the scope of this paper [15, 16]. To assess performance of our system we downloaded data from the citeseer archive[1] of computer science research papers. We built a simple database that contains the document meta data as, e.g., author and affiliation.

5.1 Computing Time for Shortest Paths

To determine document contexts we implemented the most promising main-memory based shortest-paths algorithms. In Figure 3 we compare the restricted Hidden Path algorithm with the Dijkstra algorithm. As described in Section 4, the Dijkstra computes the restricted shortest paths for document nodes only. The best results we obtained with the restricted HiddenPath algorithm.

5.2 Query Time

In Figure 4, we show the query time for different graph sizes. The values denote the mean time of 20 different queries with varying number of keywords, which

[1] `http://citeseer.ist.psu.edu/oai.html`

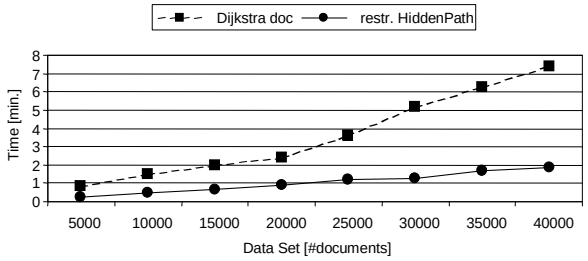

Figure 3: Running time of shortest path algorithms.

were processed subsequently. We compare the query time for two different index layouts, that we term "one-step" and "two-step" index with the query time on the pure text index. Although the queries on both context indexes run slower than on the pure text index, they are clearly fast enough to run as an interactive process. Where the single-step index shows a factor of about 2.7, the smaller two-step index yields an overhead factor of 19. In our tests, each single response time was well below the 0.5-second threshold.

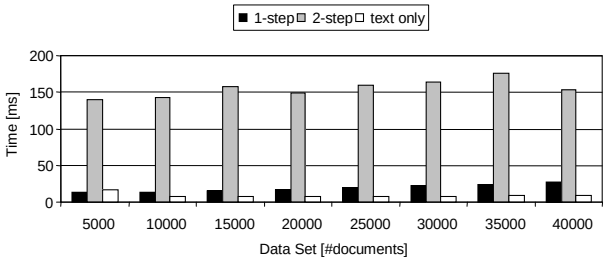

Figure 4: Query time.

6 Conclusion

In this paper, we introduced our approach to exploit database information to improve intranet document retrieval. We showed how to derive a graph structure from databases and how to embed documents inside the graph. Then, we discussed how to compute document context from this graph. Our performance experiments show that only little overhead is introduced by exploiting document context.

References

[1] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In *ICDE'02*, page 5, 2002.

[2] S. Amer-Yahia, N. Koudas, A. Marian, D. Srivastava, and D. Toman. Structure and content scoring for XML. In *VLDB'05*, pages 361–372, 2005.

[3] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE'02*, page 431, 2002.

[4] A. Burton-Jones, V. C. Storey, V. Sugumaran, and S. Purao. A heuristic-based methodology for semantic augmentation of user queries on the web. In *Intl. Conf. on Conceptual Modeling, ER'03*, pages 476–489, 2003.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. MIT Press, 1990.

[6] J. Davies and R. Weeks. QuizRDF: Search technology for the semantic web. In *Hawaii Intl. Conf. on System Sciences (HICSS)*, pages 112–119, 2004.

[7] S. Dessloch and N. Mattos. Integrating SQL databases with content-specific search engines. In *VLDB'97*, pages 528–537, 1997.

[8] N. Fuhr and K. Grossjohann. XIRQL: a query language for information retrieval in XML documents. In *SIGIR'01*, pages 172–180, New York, NY, USA, 2001. ACM Press.

[9] R. Goldman, N. Shivakumar, S. Venkatasubramanian, and H. Garcia-Molina. Proximity search in databases. In *VLDB'98*, pages 26–37, 1998.

[10] R. Goldman and J. Widom. WSQ/DSQ: a practical approach for combined querying of databases and the web. In *SIGMOD'00*, pages 285–296, 2000.

[11] J. Graupmann, R. Schenkel, and G. Weikum. The SphereSearch engine for unified ranked retrieval of heterogeneous XML and web documents. In *VLDB'05*, pages 529–540. VLDB Endowment, 2005.

[12] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-style keyword search over relational databases. In *VLDB'03*, pages 850–861, 2003.

[13] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *VLDB'02*, pages 670–681, 2002.

[14] D. R. Karger, D. Koller, and S. J. Phillips. Finding the hidden path: Time bounds for all-pairs shortest path. *SIAM Journal on Computing*, 22(6):1199–1217, 12 1993.

[15] C. Mangold, H. Schwarz, and B. Mitschang. Documents meet databases: A system for intranet search. In *COMAD'06*, 2006.

[16] C. Mangold, H. Schwarz, and B. Mitschang. u38: A framework for database-supported enterprise document-retrieval. In *IDEAS'06*, 2006.

[17] C. Rocha. A hybrid approach for searching in the semantic web. In *WWW'04*, pages 374–383, 2004.

[18] P. Roy, M. K. Mohania, B. Bamba, and S. Raman. Towards automatic association of relevant unstructured content with structured query results. In *CIKM'05*, pages 405–412, 2005.

[19] N. Stojanovic. On analysing query ambiguity for query refinement: The librarian agent approach. In *Intl. Conf. on Conceptual Modeling, ER'03*, pages 490–505, 2003.

[20] C. Yu, H. V. Jagadish, and D. R. Radev. Querying XML using structures and keywords in timber. In *SIGIR'03*, pages 463–463, New York, NY, USA, 2003. ACM Press.