Web Semantics: Science, Services and Agents on the World Wide Web xxx (2011) xxx–xxx

Contents lists available at SciVerse ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web

journal homepage: http://www.elsevier.com/locate/websem

Lightweight integration of IR and DB for scalable hybrid search with integrated ranking support

Haofen Wang [a,*], Thanh Tran [b], Chang Liu [a], Linyun Fu [a]

[a] Shanghai Jiao Tong University, Shanghai 200240, China
[b] Institute AIFB, Universität Karlsruhe, D-76128 Karlsruhe, Germany

ARTICLE INFO

Article history:
Available online xxxx

Keywords:
IR and DB integration
Hybrid search
Scalable query processing
Inverted index
Ranking

ABSTRACT

The Web contains a large amount of documents and an increasing quantity of structured data in the form of RDF triples. Many of these triples are annotations associated with documents. While structured queries constitute the principal means to retrieve structured data, keyword queries are typically used for document retrieval. Clearly, a form of hybrid search that seamlessly integrates these formalisms to query both textual and structured data can address more complex information needs. However, hybrid search on the large scale Web environment faces several challenges. First, there is a need for repositories that can store and index a large amount of semantic data as well as textual data in documents, and manage them in an integrated way. Second, methods for hybrid query answering are needed to exploit the data from such an integrated repository. These methods should be fast and scalable, and in particular, they shall support flexible ranking schemes to return not all but only the most relevant results. In this paper, we present CE^2, an integrated solution that leverages mature information retrieval and database technologies to support large scale hybrid search. For scalable and integrated management of data, CE^2 integrates off-the-shelf database solutions with inverted indexes. Efficient hybrid query processing is supported through novel data structures and algorithms which allow advanced ranking schemes to be tightly integrated. Furthermore, a concrete ranking scheme is proposed to take features from both textual and structured data into account. Experiments conducted on DBpedia and Wikipedia show that CE^2 can provide good performance in terms of both effectiveness and efficiency.

1. Introduction

Recently, we have seen a strong increase in the availability of structured data on the Web, RDF in particular. This structured data might be associated with documents in the form of annotations or might be embedded in documents. The Web as it is today can be considered as a large repository of interlinked documents, structured data and annotations. Examples of publicly available datasets on the Web that contain these different types of data are Wikipedia[1] (textual data), DBPedia[2] (annotations and domain-independent structured data) and DBLP[3] (annotations and structured data in the bibliographic domain).

Currently, keyword search is commonly supported by commercial search engines for the retrieval of documents. Beyond document retrieval, structured data on the Web can support other retrieval scenarios. Instead of documents, a search engine might return exact answers to the users as the result of a complex question, represented as a structured query. With the increase of annotations available on the Web, there is a potential to support more expressive document retrieval to address more complex needs. As discussed in [33], annotations can be seen as an additional layer of information on top of the documents, which can be exploited to answer more complex queries. In addition, annotations in form of RDFa[4] and Microformats[5] (data embedded in Web documents) are quite popular and have been used in search engines like Yahoo to improve retrieval accuracy [28].

Keyword queries and *structured queries* are the two principal means to find resources. More specialized systems such as Digital Library applications go further and employ *hybrid queries* combining both the advantages of structured queries and keyword queries. This type of queries is suitable for *hybrid search* [6], a paradigm that allows querying over textual and structured data in an integrated way. With hybrid search, the user can ask for documents or structured data, using both keywords and structural constraints. For

* Corresponding author. Tel.: +86 21 54745879 603.
E-mail addresses: whfcarter@apex.sjtu.edu.cn (H. Wang), ducthanh.tran@kit.edu (T. Tran), liuchang@apex.sjtu.edu.cn (C. Liu), fulinyun@apex.sjtu.edu.cn (L. Fu).

[1] http://wikipedia.org.
[2] http://dbpedia.org.
[3] http://dblp.uni-trier.de/.
[4] www.w3.org/TR/xhtml-rdfa-primer/.
[5] http://microformats.org/.

1570-8268/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.websem.2011.08.002

example, a user can ask for pieces of data with descriptions containing a given keyword, e.g., "Find Turing Award Winners working at IBM that are associated with documents containing *Algorithm*".

Hybrid search on a Web scale environment however brings several challenges. It requires the capacity to *store* and *index* a large amount of textual and structured data. In order to answer user queries against this data, it requires scalable solutions for integrated *processing of hybrid queries* so that results can be returned within a reasonable amount of time. Another crucial aspect is *ranking* because given the volume and the heterogeneity of Web resources, it is likely that a query results in a large number of candidates, which may differ in many aspects, including quality and recentness. Ranking is thus needed to help users focusing on the most relevant results.

In this paper, we elaborate on infrastructure components that are necessary to support large scale hybrid search. This work specifically addresses the above challenges and the main contributions are listed as follows:

- We describe a unified framework to represent and to query documents and graph-structured data in an integrated way.
- We leverage mature information retrieval and database technologies to build a repository, which can scale over a large amount of documents and graph-structured data.
- We propose a novel data structure called *Occurrence Probability Table* (*OPT*) and on top of the data structure, a set of algorithms for hybrid query processing that allows a flexible integration of advanced ranking schemes.
- We elaborate on a concrete ranking scheme which propagates and aggregates scores along a data structure called *answer tree*.

The repository and the hybrid query engine implementing our approach are embedded into an integrated solution called CE^2. We have conducted experiments with CE^2 on RDF data contained in DBpedia and on documents from Wikipedia. Results show that CE^2 supports effective ranking and scales to millions of documents and RDF triples.

The rest of this paper is organized as follows: Section 2 presents a formal model of hybrid search including definitions of resources, queries, answers and ranking. Section 3 introduces the architecture of CE^2. In Section 4 and Section 5, we elaborate on data storage and hybrid query processing in details. We show our experimental results in Section 6. Related work is presented in Section 7 and conclusions in Section 8.

2. Hybrid search

In this section, we will present a formal model of hybrid search and elaborate on its components.

Definition 1. A hybrid search model is a quadruple $\langle G, Q, F, R(q_i,d_j)\rangle$ where

(1) G is a representation of resources.
(2) Q is a representation of user information needs (queries).
(3) F is a framework that models relationships between resources and queries. Given a query, F defines which resources constitute the answers.
(4) $R(q_i,d_j)$ is a ranking function defined by $R(q_i,d_j) \in (0,1]$ *iff* d_j is an answer to q_i and $R(q_i,d_j) = 0$ otherwise.

2.1. Resources

Resources in the context of hybrid search are represented through a graph based model (*resource graph*) that contains entities, documents, their relations and attributes.

Definition 2. A *resource graph* $g \in G$ is a tuple (V,L,E) where

- V is a finite set of *vertices*, namely the disjoint union $V_E \uplus V_D \uplus V_C \uplus \{V_{Doc}\} \uplus V_L \uplus V_{Text}$ with V_E representing entities, V_D representing documents, V_C representing entity classes, V_{Doc} representing the document class, V_L representing literals, and V_{Text} representing texts.
- L is a finite set of *edge labels*, namely the disjoint union $L_R \uplus L_A \uplus \{type, subclass, annotation, keyword\}$ standing for inter-entity edges L_R and entity-attribute edges L_A.
- E is a finite set of *edges* of the structure $e(v_1,v_2)$ with $v_1, v_2 \in V$ and $e \in L$. Moreover, the following restrictions apply:
  - $e \in L_R$ if and only if $v_1, v_2 \in V_E$,
  - $e \in L_A$ if and only if $v_1 \in (V_E \uplus V_D)$ and $v_2 \in V_L$,
  - e = keyword if and only if $v_1 \in (V_E \uplus V_D)$ and $v_2 \in V_{Text}$,
  - e = type if and only if $v_1 \in V_E$ and $v_2 \in V_C$, or $v_1 \in V_D$ and $v_2 = V_{Doc}$,
  - e = subclass if and only if $v_1, v_2 \in V_C$,
  - e = annotation if and only if $v_1 \in V_D$ and $v_2 \in (V_E \uplus V_C)$ or v_2 is a resource graph.

Vertices denoting entities and documents, are labeled by identifiers, e.g., Uniform Resource Identifiers (URIs). As labels for other elements, we use class URIs, relation URIs, attribute URIs, literals and texts.

In our definition, the special vocabulary `type` represents edges that link an entity with its class or a document with the document class, `subclass` denotes a sub-class relationship between two entity classes, `keyword` represents a special attribute associating a document with its text content or an entity with some textual descriptions and `annotation` captures the relation between a document and other entities. We distinguish entity annotations ($v_2 \in V_E$) from class annotations ($v_2 \in V_C$) and from complex annotations expressed in terms of triples.

The presented data model is a conceptual representation of documents and graph-structured data on the Web. It is slightly adapted from RDF(S)[6] to suit the needs of hybrid search.

An example is given in Fig. 1(a), illustrating how structured data about Turing Award winners from DBpedia is associated with documents about algorithms and theorems from Wikipedia. Clearly, the example shows that vertices can be documents (like "Cook-levin theorem"), entities (like "Richard Karp"), entity classes (like "Living People"), the particular class identifying documents ("Doc"), literals (like "Karp") or the textual content of a document.

2.2. Queries

In order to query resources in a resource graph $g \in G$, we extend the notion of conjunctive queries defined in [21] as follows:

Definition 3. A hybrid query $q \in Q$ is an expression of the form $(x_1,\ldots, x_k).\ \exists x_{k+1},\ldots,x_m.\ A_1 \wedge \ldots \wedge A_r$, where $x_1,\ldots,x_k$ are called distinguished variables, $x_{k+1},\ldots,x_m$ are called undistinguished variables, and $A_1,\ldots,A_r$ are query atoms. These atoms are of the form $p(v_1,v_2)$, where $p \in L\backslash\{subclass\}$ is called predicate, and v_1, v_2 are called variables or terms. If p = *keyword*, it is called keyword predicate, otherwise is referred to as structure predicate.

All distinguished variables in a query must be bound to individuals, and constitute the answer to the query. All other variables in the query are undistinguished. The values bounded to undistinguished variables will not appear in the answer.

[6] The intuitive mapping from RDF(S) to our data model is: resources correspond to entities, classes to classes, properties to either relations or attributes and literals simply to literals.

Please cite this article in press as: H. Wang et al., Lightweight integration of IR and DB for scalable hybrid search with integrated ranking support, Web Semantics: Sci. Serv. Agents World Wide Web (2011), doi:10.1016/j.websem.2011.08.002

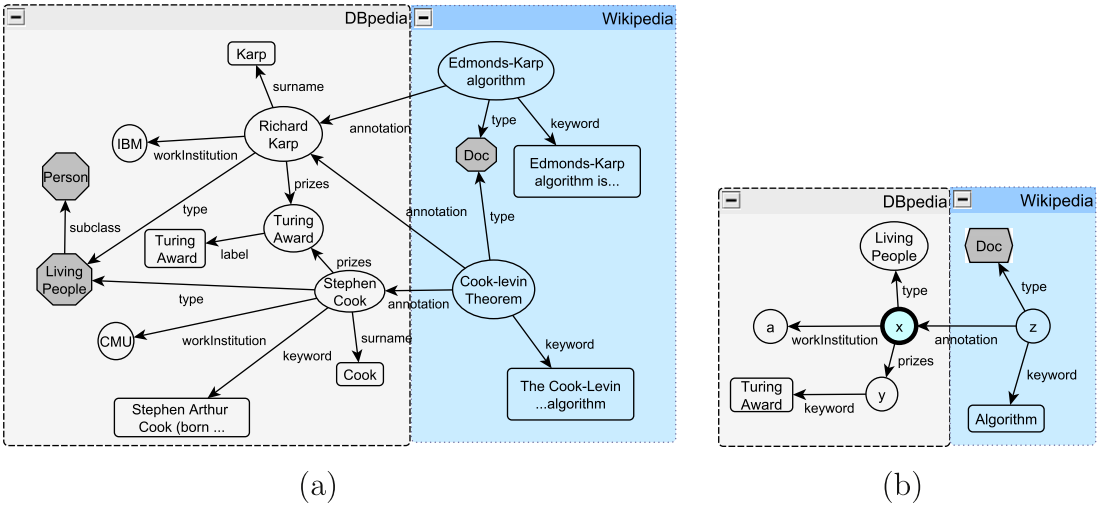

Fig. 1. Example resource and query graphs. (a) Example resource graph. (b) Example query graph.

Since variables can interact in an arbitrary way, a conjunctive query q can be seen as a graph pattern constructed from a set of triple patterns in which one or more variables might appear. In this paper, we restrict the query pattern of q to be fully-connected and tree-shaped with only one single distinguished variable (as the root). This leads to a more efficient search process [21] while still allowing a wide range of needs to be expressed [34].

Standard keyword queries on documents and structured queries on semantic data are supported as the presented hybrid query model clearly subsumes both formalisms. We now will give examples of more advanced access patterns that can be supported.

- A *hybrid entity query* can be used to retrieve entities by combining keyword and structure predicates. For instance, the query $(x).\exists y.type(x,Movie) \wedge keyword\ (x,\text{“}World\ War\text{”}) \wedge director(x,\ y) \wedge type(y,Hong\_Kong\_film\_directors)$ asks for movies made by Hong Kong Film Directors and each movie has also a textual description matching “World War”.
- An *annotation-based document query* retrieves documents by combining keyword and structure predicates on annotations. For instance, the query $(x).\exists y,z.type(x,V_{Doc}) \wedge keyword\ (x,\text{“}War\text{”}) \wedge type(y,Movie) \wedge annotation(x,\ y) \wedge director(y,z) \wedge name(z,\ \text{“}Oliver\ Stone\text{”})$ asks for documents matching “War”, and each document is annotated with movies directed by Oliver Stone.
- An *annotation-based entity query* is similar to the above one but retrieves entities contained in semantic data instead of documents. These queries are useful when the users do not have much information about the entity, but know a document that describes this entity. For instance, the user wants to search a particular movie. His best guess is that it is associated with a text, which contains the keyword “*World War*”. This can be translated to the query $(y).\exists x.type(x,V_{Doc}) \wedge keyword(x,\text{“}World\ War\text{”}) \wedge annotation(x,y) \wedge type(y,Movie)$.

An example query graph is given in Fig. 1(b). In this case, the only distinguished variable is x (highlighted). It searches for living people belonging to certain institutions, who were awarded with the Turing prize and are associated with documents containing the keyword “*algorithm*”.

2.3. Answers

We define F in our hybrid search model as follows: a solution to q on a resource graph g is a mapping μ from the variables in the query to vertices in g such that substitutions of variables in the graph pattern would yield a subgraph of g. The *substitutions* of *distinguished variables* constitute the query answers.

Definition 4. Given a resource graph $g = (V,L,E)$ and a conjunctive query q, let Var_d (resp. Var_u) denote the set of distinguished (resp. undistinguished) variables occurring in q. Then a mapping μ: $Var_d \to V$ from the query's distinguished variables to the vertices of g will be called an *answer* to q, if there is a mapping v:$Var_u \to V$ from q's undistinguished variables to the vertices of g such that the function

$$\mu' : Var_d \cup Var_u \cup V \to V \begin{cases} v \mapsto \mu(v) & \text{if} \quad v \in Var_d \\ v \mapsto v(v) & \text{if} \quad v \in Var_u \\ v \mapsto v & \text{if} \quad v \in V \end{cases}$$

satisfies $p(\mu'(v_1),\ \mu'(v_2)) \in E$ for any query atom $p(v_1, v_2)$ contained in q.

Considering the example of Fig. 1, a can be successfully mapped to “IBM”, z to “Edmonds-Karp algorithm”, y to “Turing Award” and the distinguished variable x to “Richard Karp”. Thus, “Richard Karp” is a possible answer. Similarly, constants are mapped to vertices of the resource graph. Note that the mapping of keywords to resource vertices might be imprecise. In our example, the keyword “*Algorithm*” might match the textual content of several documents, albeit with different degrees of matching. Thus, such mappings are associated with a score reflecting their degree of matching.

2.4. Ranking

In this section, we elaborate on R, the ranking scheme of our hybrid search model. In IR, the rank of a document basically measures the degree to which a document matches a given keyword query. In different IR approaches, this notion of matching incorporates different factors, e.g., TF-IDF [31], PageRank [11], etc. In databases that support keyword search on textual columns [10,8], a similar principle is used to rank tuples. Since keyword predicates can be combined with structural components, the queries supported by these databases are in fact hybrid queries. However, while these systems simply use the IR matching score (or a linear combination of matching scores obtained from several keyword predicates), we will propose a more advanced ranking scheme.

In particular, the graph structure of our data model is incorporated into the computation of final scores for the answers to a hybrid query. As discussed before, answers are bindings to distinguished variables. More precisely, they are bindings to the root node of a tree-shaped conjunctive query. Bindings to every

Please cite this article in press as: H. Wang et al., Lightweight integration of IR and DB for scalable hybrid search with integrated ranking support, Web Semantics: Sci. Serv. Agents World Wide Web (2011), doi:10.1016/j.websem.2011.08.002

node of the query tree (called elements of an *answer tree*) might be associated with a local score. The scheme we propose here is based on the propagation and aggregation of these local scores along the answer tree to arrive at a combined score for one final answer (i.e., a binding to the root node). In the following sections, we formally define the answer tree and discuss the propagation and aggregation mechanisms in details.

2.4.1. Answer tree

The evaluation of a query q results in a set of answers Ans_q. Each $ans \in Ans_q$ is of the form $(v_1, v_2, \ldots, v_k)$, where each v_i represents a binding to the corresponding variable x_i and k is the number of distinguished variables in q. Since we focus on queries with a single distinguished variable, *ans* contains one element v, which is a binding to the root node of the query tree. This element v combined with bindings to non-distinguished variables, query constants and query predicates constitute an answer tree defined as follows:

Definition 5. An answer tree g_{ans} is a tuple (V_{ans}, E_{ans}) where V_{ans} is a finite set of vertices and E_{ans} is a finite set of edges of the structure $e(v_{a_1}, v_{a_2})$ where $v_{a_i} \in V_{ans}$. Note that g_{ans} corresponds to its query tree in that they have the same structure, i.e., for each $e(v_{a_1}, v_{a_2})$, there is a corresponding query predicate $p(v_{q_1}, v_{q_2})$ such that $e = p$ and $v_{a_i} \in \mu'(v_{q_i})$ (variable bindings) or $v_{a_i} = v_{q_i}$ (constants).

Fig. 2 shows two answer trees that can be computed from the resource graph in Fig. 1(a) for the query depicted in Fig. 1(b). Besides the Ans_q = {*Richard Karp*, *Stephen Cook*}, either tree also contains query elements and bindings to other query variables. The combined information tells that Richard Karp and Stephen Cook are Turing Award winners, they work for IBM and CMU, respectively, and are associated with documents. While Richard Karp is associated with both *Edmonds-Karp algorithm* and *Cook-levin Theorem*, Stephen Cook is associated with the latter only.

2.4.2. Ranking principles

The answer tree can be regarded as the context of a given answer. Based on this context, we formulate the following two principles for ranking in hybrid search.

- *Quality propagation: the* score associated with an element of an answer tree can be seen as a measure of the quality of this element. In quality propagation, the score is updated to reflect the quality of its neighbor elements. Neighboring elements here include also those ones that are only indirectly connected (i.e., connected via a path). Since the query is always fully-connected, the answer tree is correspondingly, always a connected component. Thus, all the other elements of the answer tree represent neighbors. As a result of quality propagation, elements are assigned a higher rank if they are connected with higher-quality neighbors. This principle corresponds to the standard linear aggregation of scores, which have been computed for the different keyword predicates of a query (e.g., applied in engines such as TopX [32]). Instead of computing the aggregated score for a result tuple, we propagate the score along the answer tree to obtain the aggregated score for a binding to the distinguished variable at the root, albeit using the same principle.
- *Quantity aggregation: in* addition to the qualities of elements and going beyond standard ranking scheme, we consider also the number of neighbors. We propose quantity aggregation to incorporate this effect on the score of a given element. In quantity aggregation, the scores of all neighbors are taken into account such that elements are ranked higher when they have a larger number of neighbors.

Note that monotonicity is preserved in both principles: the higher the quality of a neighbor and the higher the number of its neighbors, the higher is the rank of an element.

In our example, Richard Karp and Stephen Cook have the same rank when only quality propagation is considered. We have only one score in this setting, which is the matching score of the keyword predicate. In this case, "Algorithm" matches in the same degree to the documents that are associated with these two person. The scores of these document nodes will be propagated to the score of the nodes representing Richard Karp and Stephen Cook. However, when quality propagation is considered in combination with quantity aggregation, Richard Karp will be ranked higher than Stephen Cook because he is associated with two documents. While the score of Stephen Cook reflects only the scores for *Cook-levin Theorem*, the score of Edmonds Karp incorporates both the scores of *Edmonds-Karp algorithm* and that of *Cook-levin Theorem*.

We will now elaborate on the mechanism for supporting these principles in the following two sections.

2.4.3. Local score assignment

An element of the answer tree, $e \in (V_{ans} \cup E_{ans})$, might be associated with a *local score* s_l. In hybrid search, s_l of an $e \in V_{ans}$ equals $RSV(e,k)$ if $e \in \mu'(x)$, where x is the variable and k is the keyword of a query predicate $keyword(x,k)$; otherwise $s_l(e)$ equals 1. $RSV(e,k)$ denotes the score for e as obtained from the evaluation of $keyword(x, k)$. The semantics of this score depends on the approach implemented by the IR engine. In line with most IR models, we choose a probabilistic semantics. That is, the score represents the probability that the textual description of e entails k. The local score s_l of an $e \in E_{ans}$ is assumed to be 1.

In our example answer tree, local scores are thus given only for two elements, i.e., the mapping results *Cook-levin Theorem* and *Edmonds-Karp algorithm* to μ'(*Algorithm*), where the scores returned by the IR engine are 0.9 for both.

2.4.4. Final score computation

The local score is different from the final score which is produced by applying two ranking principles. Namely, the final score for an answer is obtained through iterative aggregation and propagation along the answer tree, which starts from the leaf vertices and ends at the root. Here, each neighbor has a certain contribution

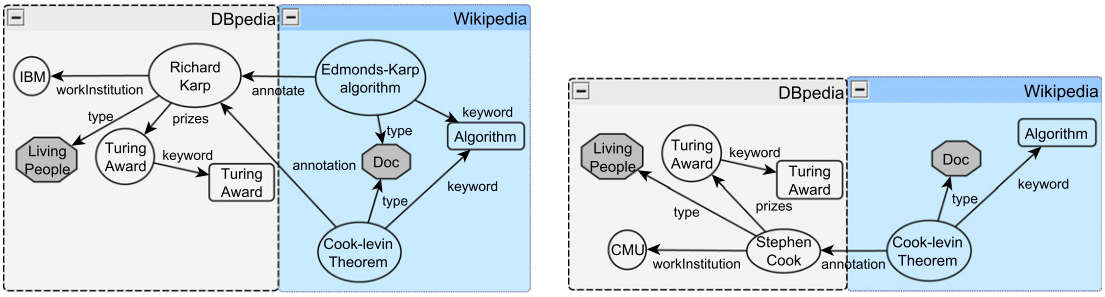

Fig. 2. Two example answer trees.

to the final score of an element. For every element, contributions are actually computed in two ways: (1) contributions of neighbors connected via the same edge type and (2) contributions aggregated over different edge types. These contributions are then propagated to the element.

In particular, the first contribution is defined as $s_c(v,e) = \perp(s(v_{n_1}), s(v_{n_2}), \ldots, s(v_{n_i}))$ where each v_{n_i} is an incoming neighbor vertex of v in V_{ans} satisfying $e(v_{n_i}, v) \in E_{ans}$. While $s_c(v,e)$ aggregates neighbor contributions with respect to a particular edge type e, the second contribution aggregates over all edge types, i.e., $s_c(v) = \top(s_c(v,e_1), s_c(v,e_2), \ldots, s_c(v,e_m))$ where e_m denotes the different types of edges associated with v.

The final score for an element can be obtained by combining the contributions as computed previously with the local score, i.e., $s(v) = \top(s_l(v), s_c(v))$ if $s_c(v) > 0$, otherwise $s(v) = s_l(v)$.

To choose the appropriate aggregation functions, several candidates are taken into account:

- As for $\top$: t-norms [25] such as
  (Minimum t-norm) $\top_{min}(a,b) = \min\{a,b\}$
  (Product t-norm) $\top_{prod}(a,b) = a \cdot b$
  (Łukasiewicz t-norm) $\top_{Luk}(a,b) = \max\{0, a + b - 1\}$

- As for $\perp$: t-conorms [25] such as
  (Maximum t-conorm) $\perp_{max}(a,b) = \max\{a,b\}$
  (Probabilistic sum) $\perp_{sum}(a,b) = 1 - (1-a) \cdot (1-b)$
  (Bounded sum) $\perp_{Luk}(a,b) = \min\{a + b, 1\}$

In line with the probabilistic semantics of the local score, $\top_{prod}$ and $\perp_{sum}$ are used throughout the paper (and for the evaluation experiments).

Another reason for not using the other aggregation functions is because they have some well-known problems when applied to search as reported in [37]. $\top_{min}$ fails to distinguish the relevance of two results r_1, r_2 when $s_l(r_1) = 0.1$, $s_c(r_1) = 0.1$, $s_l(r_2) = 1.0$, and $s_c(r_2) = 0.1$. $\top_{Luk}$ will discard any relevant answer when the sum of its local score and global score is below 1.0. $\perp_{max}$ cannot recognize the more relevant one from r_1 and r_2 when r_1 has three neighbors with scores 0.6, 0.5 and 0.4 while r_2 has two neighbors whose scores are 0.6 and 0.5. $\perp_{Luk}$ has similar problems as $\perp_{max}$.

With respect to our example, the bindings to variable z represent the leaf vertices, i.e., *Cook-levin Theorem* and *Edmonds-Karp algorithm*. Since these leaf elements (by definition) do not have incoming neighbors, no contributions can be obtained. Thus, the final scores s(*Cook-levin Theorem*) and s(*Edmonds-Karp algorithm*) are simply 0.9, i.e., equals their local scores. The contribution scores s_c(*Richard Karp*, *annotation*) and s_c(*Stephen Cook*, *annotation*) can be obtained through the formulas $\perp(s$(*Cook-levin Theorem*), s(*Edmonds-Karp algorithm*)) (=0.99) and $\perp(s$(*Cook-levin Theorem*)) (=0.9), respectively. As *annotation* is the only type of edges connected with these vertices, s_c(*Stephen Cook*) = s_c(*Stephen Cook*, *annotation*) and s_c(*Richard Karp*) = s_c(*Richard Karp*, *annotation*). The final scores for the answers are s(*Stephen Cook*) = $\top(1,0.99)$ (=0.99) and s(*Richard Karp*) = $\top(1,0.9)$ (=0.9). As a result, *Stephen Cook* is ranked higher than *Richard Karp*.

3. CE^2 architecture

CE^2 is built to store, index and perform hybrid search on textual and structured data. Fig. 3 shows the decomposition of CE^2 into two main components. The first one is the repository: textual data associated with entities and documents as well as annotations are stored in separate *inverted indexes*. Structured data other than annotations is kept in a *database*. The second component is a hybrid query engine that is composed of several sub-modules. The *Query Planner* decomposes a query into several parts, which are processed by the *Atomic Query Executor*. These results are fed to the *Query Result Combiner* and finally to the *Answer Ranker*.

The overall infrastructure represents a lightweight integration of standard IR and DB technologies. In particular, for the repository, we propose to use a standard DBMS for managing structured data and a standard IR system for managing texts. Special design choices have been made to adapt these technologies to the storage and querying needs of our graph-based data model. For integrating the two repository components and for supporting hybrid queries, we additionally introduce a component for hybrid query evaluation which employs a novel data structure called Occurrence Probability Table (OPT) for combing results and their scores.

4. Data storage and index

An efficient data storage and index scheme is essential to deal with a large amount of resources. Since resources are composed of textual and structured data, it is natural to combine IR and DB technologies. While a database offers efficient storage and advanced query optimization for structured data, inverted indexes have been successfully employed to deal with a large amount of texts. Hence, we use a database to store entity resources, namely triples of the form $e(v_1, v_2)$, where $e \in L \backslash \{keyword, annotation\}$. For textual data associated with entities and documents, namely triples of the form $keyword(v_1, v_2)$, we employ two inverted indexes referred to as *EntIdx* and *DocIdx*. Another index called *AnntIdx* is used to store annotations of the form $annotation(v_1, v_2)$. Currently, we use DB2 as the DB engine while the inverted indexes are managed by Lucene. Now we elaborate on the main design decisions made to achieve high performance.

4.1. Table schema

Databases provide many optimization features and generate efficient plans for complex queries against structured data. Data partitioning is an important design decision that has a significant impact on the performance. So far, the most commonly used approach involves a single table, which contains all triples [17,36,29]. Recently, vertical partitioning [1] has been suggested. This approach partitions properties (correspond to edge labels $e \in L_R \uplus L_A \uplus \{type, subclass\}$) into different tables such that n tables are created for n distinct properties. In this way, each table contains only triples for a given property, which makes it more compact. In particular, it can avoid the many self-joins that would be necessary when processing queries on a single table scheme. Also, it is easier to obtain statistics for further join-order optimization. However, it is not flexible enough to handle update, especially since the cost of table creation or deletion is expensive when properties are added or removed. Furthermore, too many tables have to be created when there are too many properties, which leads to too much burden on database for a large amount of metadata that needs to be managed.

Drawing conclusions from the weaknesses and strengths of these approaches, we divide the single big table into several small ones for *type*, L_R and L_A. Precisely, we build three tables $Type(v_1 \in V_E$ as E, $l \in V_C$ as $C)$, $Relation(v_1 \in V_E$ as S, $l \in L_R$ as R, $v_2 \in V_E$ as $O)$ and $Attribute(v_1 \in V_E$ as S, $l \in L_A$ as A, $v_2 \in V_L$ as $L)$. Since the class hierarchy is relatively small in size, we simply store it in the main memory for fast access rather than build a table for *subclass*. This scheme can avoid the update and metadata management issues in vertical partitioning and at the same time can reduce the number of self joins required by the single table approach.

4.2. Database Indexes

Various kinds of indexes are commonly employed in triple stores to allow more efficient lookup. B+-tree indexes are typically

Please cite this article in press as: H. Wang et al., Lightweight integration of IR and DB for scalable hybrid search with integrated ranking support, Web Semantics: Sci. Serv. Agents World Wide Web (2011), doi:10.1016/j.websem.2011.08.002

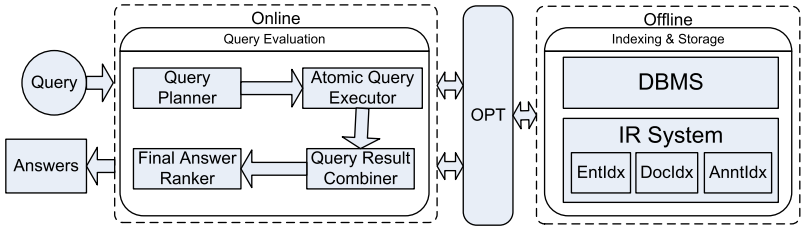

Fig. 3. Architecture of CE^2.

used for fast access on specific columns. With respect to our table schema, an index for *l* (stands for a class or a relation or an attribute) would be a natural choice as triple patterns in most queries contain a value on *l*. A table might contain many triples that are associated with *l*. Given a query asking for triples associated with a particular *l*, many lookups are required to obtain pointers. These pointers possibly refer to many blocks which have to be fetched from disk to obtain the final values.

In order to minimize disk accesses, we adopt a concept called Multi Dimensional Clustering (MDC) [9]. In particular, we employ three MDC indexes, namely $C*$ on column *C* for *Type*, $R*$ on column *R* for *Relation* and $A*$ on column *A* for *Attribute*. Using MDC this way, triples associated with a given class (relation or attribute) are stored in one physical block – or continuously stored blocks when there are many such triples. This results in better locality of access and a more optimized prefetching of blocks. MDC also involves the use of block indexes which contain entries that point to blocks. Such block-based indexes contain fewer entries and thus, a lookup on the index has lower path length overhead.

Furthermore, we consider several secondary indexes to cover as many access patterns as possible. In particular, indexes on the columns (E,C), (S,R,O) and (O,R), (S,A,L) and (L,A) are built for *Type*, *Relation* and *Attribute*, respectively. Similar combination of secondary indexes and MDC has been proposed (mainly for OWL) by [26]. This approach has achieved superior performance in a comparison with several state-of-the-art systems.

4.3. Inverted indexes

While database is a general purpose system that can handle all types of data, the inverted index has proven to be an efficient and effective solution for textual data. We employ two separate inverted indexes for keyword search on textual descriptions of entities (*EntIdx*) and documents (*DocIdx*). Instead of a database, an additional inverted index called *AnntIdx* is used for the retrieval of annotations. There are two reasons for this design.

Annotations relate documents with entities. Typically, there exist several annotations for a particular document. Thus, the number of annotations that need to be managed by a hybrid search engine is potentially large. In DBpedia for instance, the number of annotations is much larger than that of any other relation and attribute assertions. Hence, the number of index lookups and disk accesses necessary for the evaluation of *annotation* (as a predicate) is possibly high. The use of inverted indexes can alleviate this problem. In [34], an RDF store called Semplore has been proposed, which relies completely on inverted indexes for the storage and retrieval of RDF data. Compared with database solutions, Semplore exhibits better locality of access. However, database solutions are superior in handling more complex queries, mainly due to the availability of various (query) optimization techniques.

We use an indexing scheme similar to Semplore to manage annotations. However, a different method is employed to map triples to documents. While the subject of an annotation is stored as a term, the object is stored as a document containing the term. In Semplore, the relation and attribute labels are stored as terms while their subjects and objects are stored as documents and term positions in the documents, respectively. This difference has the effect that in CE^2, annotations can be directly retrieved for a given subject *s*, i.e., by submitting *s* as a keyword query. In Semplore, RDF triples can only be directly retrieved for a given relation or attribute. Thus, the entire list of annotations have to be fetched first, i.e., by submitting *annotation* as a keyword query. The returned list then needs to be scanned to find triples containing *s*. Moreover, unlike Semplore, the rest of the structured data is managed by a database in CE^2. This design aims to combine superior locality of access on annotations with the optimization features provided by the database.

More importantly, managing annotations separately allows a flexible integration of ranking into hybrid query processing.

The processing of an annotation-based hybrid query (c.f. examples in the previous section) involves the evaluation of *annotation* and *keyword* predicates, one *keyword* predicate on a document and the other on an entity description. The evaluation of the *keyword* predicates results in two answer sets containing entries with scores. These entries are documents and entities, respectively, which in a next step, are joined with the results of *annotation*. When doing this join, the scores obtained from the keyword queries have to be propagated and aggregated as discussed in Section 2.

Some databases (either relational or XML) offer keyword search on textual attributes [7,10,8], i.e., can be used to evaluate *keyword*. The ranking of final results is simply based on the scores obtained for the bindings to the *keyword* predicate. It is not straightforward to extend this in-built ranking mechanism to support the aggregation and propagation of multiple scores. One naive implementation is to perform the ranking after the query has been completely processed by the database engine. In this case, answer trees have to be reconstructed for the results obtained from the engine. This is an overhead that can be avoided when annotations are managed separately. This design moves the evaluation of annotation-based hybrid queries to the hybrid query processing layer. On this layer, different ranking scheme can be applied on the intermediate results obtained from the database and the inverted index, i.e., to perform ranking when joining results of the *keyword* predicates with annotations. Answer trees are not reconstructed for the final answers, but are maintained during hybrid query processing. Thus, it is a more flexible and efficient mechanism for integrated ranking, which can leverage standard IR and database technologies (without requiring any modifications).

4.4. Directory encoding

We assign unique IDs for both entities and documents and store entries of the form (*ID*,*URI*) in a dictionary table. A primary clustered index on *ID* is created to enable fast lookup. Another primary index is also created on *URIs* to locate IDs more quickly, given the URI. Many operations performed at query processing time such as scan and join involve entities and documents. This approach of dictionary encoding can improve efficiency since these operations can be performed on the more compact IDs. The URIs are retrieved from the dictionary table only for the final results.

Please cite this article in press as: H. Wang et al., Lightweight integration of IR and DB for scalable hybrid search with integrated ranking support, Web Semantics: Sci. Serv. Agents World Wide Web (2011), doi:10.1016/j.websem.2011.08.002

5. Query evaluation process

This section describes the whole process of query processing. At first, the hybrid query is decomposed into several sub-queries. Each sub-query is then evaluated by the corresponding atomic query executor. The returned answers and their associated scores are propagated and aggregated according to an optimized query plan, which reflects the ranking principles mentioned in Section 2. Finally, the answers are ranked on the basis of the calculated scores.

5.1. Query decomposition

At this stage, a hybrid query is decomposed into a finite set of sub-queries which might be atomic queries composed of exactly one query atom or complex queries which contain several query atoms. The sub-queries are of the following four types:

- Text query ($Q_{q_{Text}}$), containing atomic queries used to find entities whose descriptions contain certain keywords.
- Document query (Q_{q_D}), containing atomic queries used to find documents containing certain keywords.
- Annotation query ($Q_{q_{Annot}}$), containing atomic queries used to find document-entity pairs.
- Entity query (Q_{q_E}), containing queries with edges being relations (including *type*) or attributes. Such queries might be atomic or complex queries.

Thus, the decomposition of a hybrid query results in several atomic queries and possibly, several complex entity queries. Note that variables in sub-queries are treated as distinguished so that answers matching all query nodes can be obtained for further aggregation. Since entity queries can be processed directly by the database, they are treated as a special kind of atomic queries during the atomic query execution stage.

Fig. 4 shows how our example query is decomposed into four sub-queries, $subq_1$ to $subq_4$, each of a different type.

5.2. Atomic query execution

Each sub-query is processed by a specific atomic executor running either on top of the database or of the inverted indexes. The answers for each sub-query are stored in a data structure called *Occurrence Probability Table* (OPT), which is defined as follows:

Definition 6. Given a hybrid query q, an OPT_q is a tabular data structure with m columns and n rows, where m equals the number of distinguished variables in a hybrid query q. Answers are stored as rows, where each cell is a pair (d,p) associating a data value d with a score p. We denote $OPT_q[i,j]$ as the cell in the i-th row and the j-th column of OPT_q, $OPT_q[i,j].d$ and $OPT_q[i,j].p$ as the data and score field of the cell, respectively, $OPT_q.r[i]$ as the i-th row of OPT_q, and $OPT_q.c[j]$ as the j-th column, where $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$. There is special score field associated with each row denoted as $OPT_q.r[i].p_r$. It captures the aggregated score computed for an answer tuple during result combination. Further, $OPT_q.n$ denotes the number of rows and $OPT_q.m$ denotes the number of columns. $OPT_q.r$ and $OPT_q.c$ represent the row and column set, respectively.

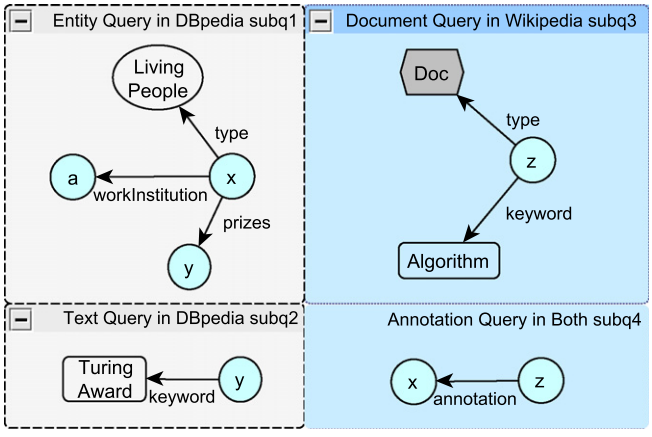

Fig. 4. Sub-queries of the example query by decomposition.

With regard to our ranking scheme, the score p of each cell can be flexibly computed using different models, and thus is open for many possible interpretations, e.g., it could be a truth value, a confidence or a measure of imprecision. In this paper, we use the score returned by the underlying IR engine for text and document queries as the value of this score.

During atomic query execution, each entity query $subq \in Q_{q_E}$ is rewritten as a set of conjunctive SQL queries. During rewriting, we use table names based on the database schema as mentioned in Section 4.1. The results are stored in a multi-column OPT. The score for each cell is set to 1.0. Taking the entity query in Fig. 4 as an example, a three-column OPT is returned, as shown in Fig. 5. Each text query $subq \in Q_{q_{Text}}$ is processed by the IR engine against the entity index *EntIdx*. The returned entities along with the matching scores fill an one-column OPT. Similarly, each document query $subq \in Q_{q_D}$ is submitted to the document index *DocIdx*, and another one-column OPT is produced. For all the OPTs produced during this step, p_r is set to 1.0. Note that annotation queries $subq \in Q_{q_{Annot}}$ are not processed during this step, as each of them would match all annotations.

The OPTs obtained for our example query are shown in Fig. 5.

5.3. Query Result Combination

In this step, the sets of OPTs previously obtained for subqueries in Q_{q_E}, Q_{q_Text} and Q_{q_D} are combined. From now on, the combining operation between two OPTs is denoted as $\oplus$. It is realized through two join operations called *Keyword Join* and *Hybrid Join*. For example, in Fig. 4, $subq_1 \oplus subq_2$ is $(x,y).starring(y,x) \wedge type(y,Film) \wedge keyword(y,\text{“}comedy\text{”})$. Query q in Fig. 1 can be obtained by $subq_1 \oplus subq_2 \oplus subq_3 \oplus subq_4$.

5.3.1. Keyword join

This operation is used to combine OPTs of an entity query q_1 and a text query q_2, with q_1 and q_2 sharing an entity variable e. As shown in Algorithm 1, it returns a combined OPT that has the same table schema like OPT_{q1}. For computing this, results obtained from the keyword search query q_2 are added to the entity information obtained from q_1. In our example, $\bowtie_{keyword}$ is needed for the combination of OPT_{subq_1} and OPT_{subq_2}. Algorithm 1 shows the details of $\bowtie_{keyword}$, where $col(OPT_q, v)$ is a helper function to locate the column in OPT_q that corresponds to the distinguished variable v in q. This is implemented as an extension of nested loop join. Since entities stored in OPT_{q2} are sorted by IDs, binary search is employed for a faster scan. When tuples of the two OPTs join on an entity, the corresponding score field is updated according to the matching score obtained from OPT_{q2}. Due to binary search, the time complexity is $O(n_1 \cdot \log(n_2))$, where n_1 and n_2 are the numbers of rows in OPT_{q1} and OPT_{q2}, respectively. Usually, OPT_{q2} is much larger than OPT_{q1} as keyword search often supports imprecise matching, resulting in a large number of answers. Thus, we iterate over the smaller OPT, and search on the larger one. Another speedup is obtained by caching results to avoid unnecessary binary searches. As shown on the top of Fig. 5, OPT_{subq_5} is returned by combining the OPTs of $subq_1$ and $subq_2$, i.e., $subq_5$ is the combination $subq_1 \oplus subq_2$.

Please cite this article in press as: H. Wang et al., Lightweight integration of IR and DB for scalable hybrid search with integrated ranking support, Web Semantics: Sci. Serv. Agents World Wide Web (2011), doi:10.1016/j.websem.2011.08.002

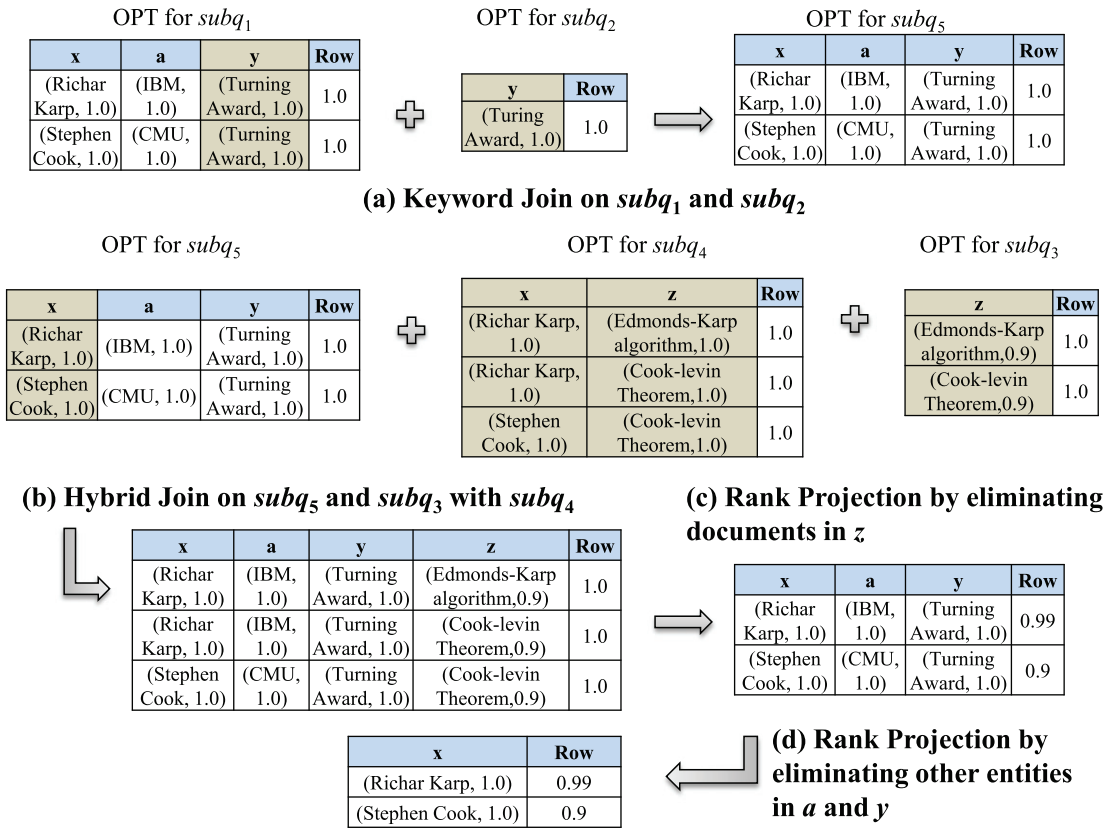

Fig. 5. Result combination and ranking on the sub-queries of the example query.

Algorithm 1

$\bowtie_{keyword}(\text{OPT}_{q1}, \text{OPT}_{q2}, e)$

$q^* := q_1 \oplus q_2$
$pos := col(\text{OPT}_{q_1}, e)$
for each row i in OPT_{q1} **do**
  $val := \text{OPT}_{q1}[i,pos].d$
  if binary search on the only column in OPT_{q2} found $\text{OPT}_{q2}[j,1].d = val$
  then
    add a new row $\text{OPT}_{q1}.r[i]$ as the k-th row in OPT_{q*}
    update $\text{OPT}_{q*}[k,pos].p$ to $\text{OPT}_{q2}[j,1].p$
  end if
end for
return OPT_{q*}

5.3.2. Hybrid join

This operation is used to combine OPTs of an entity query q_1 and a document query q_2. In fact, results of q_1 might be joined with those of a combined query $subq_2 \oplus subq_3$, where q_2 is a document query that do not have a common variable with q_1, and q_3 shares with q_1 the entity variable e and with q_2 the document variable d. In this case, an entity query q_1 is combined with a document query q_2 with the help of an annotation query q_3.

This combination returns a combined OPT whose number of columns is the sum of those in OPT_{q1} and OPT_{q2}. In our example, a join of entities and documents with the help of annotations is needed for the combination of OPT_{subq_5} and OPT_{subq_3} through OPT_{subq_4}, as shown in the middle of Fig. 5.

Algorithm 2 shows the details of $\bowtie_{hybrid}$, where $getObject(k)$ returns the documents or entities stored in O_{val} by submitting keyword k to *AnntIdx*, $find(o,c)$ locates the rows whose data fields of the c-th column all equal the given value o, and $concat(row_1,row_2)$ simply concatenates two given rows to return a new row. Since we have to iterate through all results, including the annotations obtained from the atomic queries, the time complexity is $O(n_1 \cdot |O_{val}| \cdot n_2)$, where n_1 and n_2 are the number of rows in OPT_{q1} and OPT_{q2}. When binary search is possible (e.g., q_2 is a document query), the complexity can be reduced to $O(n_1 \cdot |O_{val}| \cdot \log(n_2))$.

Algorithm 2

$\bowtie_{hybrid}(\text{OPT}_{q1}, \text{OPT}_{q2}, q_3, e, d)$

$q^* := q_1 \oplus q_3 \oplus q_2$
$pos_1 := col(\text{OPT}_{q_1}, e)$
$pos_2 := col(\text{OPT}_{q_2}, d)$
for each row i in OPT_{q1} **do**
  $val := \text{OPT}_{q1}[i,pos_1].d$
  $O_{val} := getObject(val)$
  for all $o \in O_{val}$ **do**
    $M := find(o, \text{OPT}_{q_2}.c[pos_2])$
    for all $m \in M$ **do**
      $r := concat(\text{OPT}_{q_1}.r[i], \text{OPT}_{q_2}.r[m])$
      add r as a row in OPT_{q*}
    end for
  end for
end for
return OPT_{q*}

For join order optimization in result combination, we make use of two heuristics. First, according to the analysis of both join operations, we find that $\bowtie_{keyword}$ in fact, acts as a filter making the combined OPT much smaller than the original OPTs while $\bowtie_{hybrid}$ actually expands the OPTs resulting in a combined OPT with much

larger size. Thus, $\bowtie_{keyword}$ should be performed ahead of $\bowtie_{hybrid}$. Second, the smaller an OPT is, the earlier a join operation should be performed on it. Taking the previous query as an example, $OPT_{subq}1$ and OPT_{subq2} should be combined first via $\bowtie_{keyword}$, and then the resulting OPT_{subq5} shall be combined with OPT_{subq3} via $\bowtie_{hybrid}$ to get the final answers.

5.4. Rank Projection

Based on the OPT, which has been introduced to store elements along with their scores, we define π_{rank}, also referred to as *Rank Projection*. Given OPT_q and the set of distinguished variables X, π_{rank} returns an OPT'_q with $|X|$ columns, which contains the final answers along with scores that reflect contributions of neighbors.

Algorithm 3 shows the details of π_{rank}. Firstly, a projection on the variables in X is applied to obtain the final answers Ans_q. Then, the final score for each answer is computed according to the propagation and aggregation mechanisms. Contributions from undistinguished variables are propagated and aggregated to form the scores for distinguished variables. Columns corresponding to undistinguished variables are eliminated during this process. For this, Algorithm 3 employs many sub-procedures: $incoming(v,e)$ finds all nodes connected with v through the edge e on the query graph, regardless of e's direction. $edges(v)$ finds all the edges connecting to v. $s_c(v,e)$ is the aggregated score which is computed by propagating scores from different nodes connected with v through edge e to v, while $s_c(v)$ aggregates all the scores $s_c(v,e_i)$ that have been computed for different edges.

In particular, contributions from incoming vertices through an edge type e, i.e., $s_c(v,e)$, are computed for each element of an answer. For this, $incoming(v,e)$ mentioned in Algorithm 3 finds all incoming vertices connected with v through e. Then contributions from different types of edges are combined to obtain the aggregated contribution for an element, i.e., $s_c(v)$. The function $edges(v)$ in Algorithm 3 returns all different edge types connecting with v. These contributions are computed for all v of ans. Then, they are aggregated to obtain a combined contribution for all elements in ans, i.e., $s_c(ans)$. The final score for the answer ans is obtained by the aggregation of the local score of ans (as given in $OPT'_q.r.p_r$) and the combined contribution $s_c(ans)$.

At the bottom of Fig. 5, we show two steps of π_{rank} operations. In the first step, the contribution of documents have been aggregated and propagated to their associated entities, thus the column representing documents is removed. In the second step, we continue to "shrink" the OPT until only one column for the target variable remains. These two steps will be illustrated also in the discussion on iterative ranking computation to be presented in the following subsection.

The time complexity of π_{rank} is $O(m \cdot n + (m - |X|) \cdot n)$ where m denotes the number of columns in the input OPT, and n corresponds to the number of rows. The $m \cdot n$ part captures the cost for scanning every row in OPT_q through breadth-first search. The $(m - |X|) \cdot n$ part includes the cost for the projection, as well as the aggregation and propagation of scores. The complexity of this second part is much lower in practice because there often are many redundant values in each column. In this case, fewer computations are required for obtaining the contributions.

Algorithm 3

$\pi_{rank}(OPT_q, X)$

for each $ans \in Ans_q$ **do**
  add ans as a row r to OPT'_q
  for each v contained in ans **do**
    for each e connected with v **do**
      compute contributions $s_c(v,e) := 1 - \prod(1 - s(v_{n_i}))$, where $v_{n_i} \in incoming(v,e)$, there are totally $|incoming(v,e)|$ of them.
    end for
    compute aggregated contributions $s_c(v) := \prod s_c(v,e_i)$, where $e_i \in edges(v)$, there are totally $i = |edges(v)|$ of them
  end for
  compute aggregated contributions $s_c(ans) := \prod_{v \in ans} v$
  $OPT'_q.r.p_r := OPT'_q.r.p_r \cdot s_c(ans)$
end for
return OPT'_q

5.5. Iterative rank computation

One straightforward strategy to perform ranking is to compute the OPT for the entire query and apply π_{rank} on it. However, the size of the OPT increases dramatically with the number of variables in a query, which leads to rather expensive execution of π_{rank} for complex queries, which is exponential to the number of columns of the OPT.

Therefore, the "divide and conquer" strategy is adopted to integrate ranking into query processing. First, we use the decomposition previously discussed to split a query q into sub-queries belonging to $Q_{q_{Text}}$, Q_{q_E}, Q_{q_D}, and $Q_{q_{Annot}}$. During query result combination, text queries and entity queries are processed with $\bowtie_{keyword}$ joins. We call the combination of an entity query and a text query a *keyword-joined entity query*. This type of query and the document query are connected through an annotation query. When document queries and keyword-joined entity queries are regarded as nodes, and annotation queries are regarded as edges, they together form a tree in which the root node is the one that corresponds to the distinguished variable node of q, i.e., it is a document query or a keyword-joined entity query. Based on this tree, the plan for iterative rank computation is derived.

In particular, depth-first search is performed on the tree and scores are propagated and aggregated from the leaf nodes iteratively until reaching the root. During each iteration, π_{rank} is performed on the OPT returned by $\bowtie_{hybrid}$, resulting in a much smaller OPT with fewer columns for further operations. Finally, another π_{rank} is performed on the root node to obtain the final answers along with their scores. Due to the decomposition and the iterative processing of ranking along the tree, the performance improves significantly in most cases as the size of the OPTs to be processed is much smaller.

Let us consider the example query mentioned at the beginning of the section: $subq_5$ is the root of the tree, which is connected with $subq_3$ through the edge $subq_4$. First, a π_{rank} is performed on the OPT returned by $\bowtie_{hybrid}(OPT_{subq_5}, OPT_{subq_3}, OPT_{subq_4}, x, z)$, resulting in an OPT that shares the same variables as OPT_{subq_5}. Then another π_{rank} is performed on this OPT to get final ranked answers. The process is shown at the bottom of Fig. 5.

More precisely, we keep the combined OPT (= $\bowtie_{hybrid}(OPT_{q1}, OPT_{q2})$) with the table schema as OPT_{q1} or OPT_{q2} resulting from $\pi_{rank}(OPT_{q^*}, OPT_{q_1}.m)$ or $\pi_{rank}(OPT_{q^*}, OPT_{q_2}.m)$, respectively. This one is used for further combination with other OPTs to be processed. Thus, we can always benefit from the low time complexity of $\bowtie_{hybrid}$. For example, by performing π_{rank} on the combined OPT = $\bowtie_{hybrid}(OPT_{subq_5}, OPT_{subq_3}, subq_4)$, the resulting OPT has the same

Please cite this article in press as: H. Wang et al., Lightweight integration of IR and DB for scalable hybrid search with integrated ranking support, Web Semantics: Sci. Serv. Agents World Wide Web (2011), doi:10.1016/j.websem.2011.08.002

table schema as OPT_{subq_5}, and the row scores of this OPT represent the aggregated scores of documents in OPT_{subq_3}.

6. Experiment

We have conducted all experiments on a workstation with 4 Pentium D 3.2 GHz processors and 4GB memory, running on Sun JRE 1.5 and Microsoft Windows Server 2003. Resources comprise RDF data from DBpedia [3] and documents from Wikipedia. Together, they represent more than 4.3 million triples and 2.1 million documents, and about 42 million annotations. Textual data and annotations are maintained in inverted indexes implemented using Lucene 2.4.1[7], and the rest of the RDF data is stored in IBM DB2 V9[8]. The experiments are carried out using the following query sets in which QS1, QS2, QS3, QS4 and QS5 are used to test the efficiency while QS4 and QS5 are also used for the effectiveness evaluation:

- *QS1* is used to test the access locality on annotations. It consists of 20 annotation queries. An example is "find documents annotated with http://dbpedia.org/page/Berlin".
- *QS2* is designed to test join operations on annotations. It consists of 20 queries. Each contains two predicates, where one is `annotation` and the other is a relation or an attribute, e.g., "find documents annotated with companies".
- *QS3* contains 20 queries. The number of predicates in each of them is between 2 and 9, e.g., "find institutions of scientists that are annotated in http://en.wikipedia.org/wiki/IBM". These queries are designed to test the processing of more complex queries.
- *QS4* consists of 30 queries manually created according to the questions provided by 10 users. These queries range from simple to complex tree-shaped conjunctive queries that consist of up to seven predicates. An example is "find institutions that Turing Award winners work in". Note that "Turing Award winner" is not given explicitly in the structured data but is contained only in some entity descriptions. In contrast to queries in the previous sets, each query in this set contains at least one keyword predicate, i.e., they are hybrid queries. They are designed to evaluate the hybrid query processing capability.
- *QS5* consists of queries that are constructed using the information from the Wikipedia page "List of film director and actor collaborations" and the page "List of film director and composer collaborations". Names for directors, actors and composers listed in these pages are used as terms of two query patterns, one involves the relations "directed by" and "acted in" and the other involves the relations "directed by" and "composed by". Instantiating these query patterns with the given names results in a total of 287 hybrid entity queries which ask for films.

6.1. Effectiveness study

Using QS4, we compare the effectiveness of the ranked results produced by CE^2 against Lucene and DB2 with Text Extender[9].

We indexed Wikipedia pages in Lucene. In DB2 with Text Extender, both the pages and the RDF data from DBpedia are stored. In particular, we create two additional tables to store textual descriptions for entities and documents. Annotations and other RDF data are stored in a specific `Relation` table. Since QS4 lacks ground truth, we conducted a survey with 20 participants from our laboratory to obtain a baseline for assessing precision. Each participant was asked to rate the top 10 results for queries produced by the three systems. Each query was rated by at least three persons. Only results receiving a unanimous judgement of "relevant" are used as the ground truth. We use the standard IR metric *P@n* to compute the precision for the top *n* results returned by the systems. Note that the queries in QS4 were submitted by the users arbitrarily and no ground truth was available, so it was impossible to calculate the recall for QS4, and thus no *F*-measure scores can be returned. You can refer to http://www.apexlab.org/apex_wiki/QS4Detail for the complete query list of QS4 and the detailed effectiveness performance of each query with respect to *P@n* is also reported.

Fig. 6(a) shows that search on both structured and textual data can greatly improve precision, compared with keyword-based search. In particular, the *P*@10 of DB2 with Text Extender is about 20 percent higher than that of Lucene for the query "find institutions that Turing Award winners work at". To answer this query, DB2 with Text Extender can exploit structured data about institutions and persons in DBpedia and combine it with text about Turing Award winners in Wikipedia. In contrast, Lucene entirely relies on documents and returns results simply containing the terms "institution", "Turing Award winner" and "work".

However, the ranking mechanism in DB2 with Text Extender – and other databases with support for IR-style queries on textual data – only considers the scores for keyword search on the text columns. In this case, only the scores obtained from the imprecise matching of predicates on textual attributes are incorporated. CE^2 takes this further to consider propagation and aggregation of scores along elements of the query graph. With respect to the example query, institutions are ranked higher by CE^2 when they are related with a large number of persons with textual descriptions that strongly match "Turing Award winners". The matching scores obtained for the keyword predicate is propagated and aggregated along the answer tree. In DB2 with Text Extender, the score is simply the matching score while effects of neighbors are neglected. In Fig. 6(a), the average performance computed for the 30 queries in QS4 is illustrated. Clearly, the average performance of CE^2 is superior to that of Lucene as well as that of DB2 with Text Extender. The result thus suggests that search is more effective when contributions of elements in the answer graph are considered for ranking.

We have carried out another experiment using the ground-truth as provided in the Wikipedia pages. In particular, ground-truth is established by the films that are listed in the Wikipedia pages as discussed for QS5. Note that queries in QS5 are automatically created using the directors, actors and, respectively, composers that are listed together with the films. The average precision and recall of CE^2 for QS5 are shown in Table 1. "List of film director and actor collaborations" are 0.83 and 0.71 while it achieves 0.74 and 0.64 as the precision and recall performance for the other 60 queries. We also list the overall *F*-measure scores for the two subsets of queries in QS5 (i.e., 0.765 for director + actor and 0.686 for director + composer) in Table 1. These results indicate that hybrid search can be supported effectively by CE^2. In fact, precision and recall might be even higher when existing problems with Wikipedia and DBpedia such as non-English titles and inconsistent dataset versions are resolved.

6.2. Efficiency of Annotation Management

We compare CE^2 with *Semplore* where annotations are treated just like other structured data, i.e., it is stored in one inverted index. Additionally, we have experimented with two alternative database solutions where annotations are managed differently. In particular, we have employed two different schemas for the storage of annotations using DB2. The first one consists of a *single table* which stores all RDF triples in *Relation*. The second one is a *binary table* scheme which separates annotations from the remaining RDF

[7] http://lucene.apache.org/.
[8] http://www-306.ibm.com/software/data/db2/9/.
[9] http://www-306.ibm.com/software/data/db2/support/textextender/.

Please cite this article in press as: H. Wang et al., Lightweight integration of IR and DB for scalable hybrid search with integrated ranking support, Web Semantics: Sci. Serv. Agents World Wide Web (2011), doi:10.1016/j.websem.2011.08.002

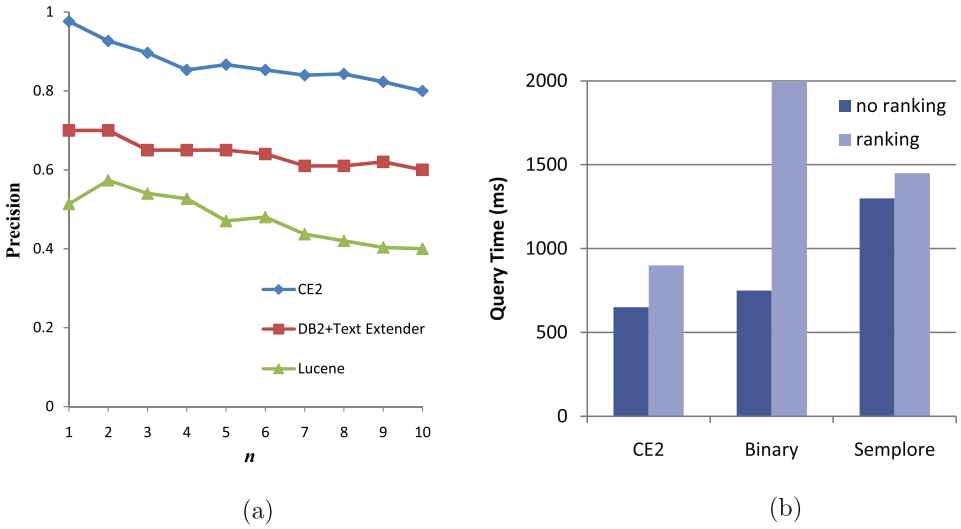

Fig. 6. (a) Precision at n for QS4. (b) Average response time (ms) for QS4.

Table 1
Effectiveness performance for QS5.

	No. of queries	Precision	Recall	*F*-measure
Director + actor	227	0.83	0.71	0.765
Director + composer	60	0.74	0.64	0.686

triples. For all tables, we have built indexes on the "subject-object" columns and the "object" column, respectively.

The average response time for CE^2 and the alternative solutions are shown in Fig. 7. for QS1, QS2 and QS3. The vertical scale is logarithmic to incorporate the wide range of response times that have been obtained. QS1 tests whether a given scheme provides fast access to annotations. QS2 tests how efficient joins including annotations can be processed. QS3 targets the join-order strategy for query optimization. We can see that the single table database has the worst performance, except for QS1 where it is better then Semplore. To process QS1, Semplore scans all annotations to locate the entries matching the constant specified in the query. This is different from the databases, which can make use of indexes for fast lookup.

Due to a special mapping of annotations to terms and documents in an inverted index, CE^2 can directly retrieve all annotations for a given constant, just like using an index in the database. Moreover, these annotations are placed contiguously on the disk and can thus be accessed very fast. Accordingly, CE^2 provides the fastest response time for queries in QS1. With respect to QS2, Semplore performs better than both database solutions. Since many objects have to be retrieved to answer QS2, many index lookups are required. In this case, a scan as performed in Semplore can be more efficient. For QS3, the binary table database outperforms Semplore. This is partly due to the effect of query optimization, which is more important when dealing with complex queries. Since CE^2 can leverage the optimization capability of the underlying database for querying structured data, it also offers the best performance for QS2 and QS3. Compared with the database solutions, CE^2 benefits from access locality when querying annotations.

6.3. Efficiency of ranking

The previous efficiency evaluation has been carried out with non-hybrid queries, i.e., no ranking was involved. Using QS4, we will now discuss the impact of ranking on performance. While hybrid search has attracted much attention, Semplore [34] is the only engine we found that is capable of handling hybrid queries. To design a comparative evaluation that is fair and conclusive, we implemented ranking on top of the binary table database used for the previous evaluation by (1) marking all variables as distinguished and submitting the query against the database, (2) reconstructing the answer tree from the answers and the query elements, and (3) propagating and aggregating scores on the answer tree. Furthermore, we let both Semplore and the binary table database use the same ranking scheme as CE^2.

As shown in Fig. 6(b), Semplore seems to be more efficient in ranking computation than the binary table database. This is because ranking is performed during join processing while in the database, ranking is performed only after query processing. Thus, ranking in the database suffers from the expensive calculation on the query graph when the final answer set is large. Clearly, the database is superior to Semplore when ranking is not involved. This is mainly due to the query optimization, which is not supported in Semplore.

Compared with these two extremes, CE^2 seems to be superior both in no-ranking and ranking modes, especially in the latter. CE^2 integrates ranking into query processing, while still leveraging query optimization of the underlying database. It benefits from database optimization when answering complex subqueries on semantic data. Ranking is performed on the OPTs obtained from these subqueries. Thus, it is performed during query evaluation and does not require a re-construction of the answer graph.

7. Related work

There exist several categories of related work. We structure our discussion to successively cover the following aspects: (1) hybrid search model, (2) storage of RDF data, (3) IR and DB integration, and (4) ranking.

Hybrid search model – recently, search involving the combination of text, structured data and annotations has attracted much attention. In [18], a logical framework has been proposed to support a wide range of queries over annotations. The use of annotations for document retrieval has also been elaborated in [13]. The authors of this work proposed an adoption of the vector space model for information retrieval where instead of using terms, annotations become the basic unit of information. These annotations are in fact

Please cite this article in press as: H. Wang et al., Lightweight integration of IR and DB for scalable hybrid search with integrated ranking support, Web Semantics: Sci. Serv. Agents World Wide Web (2011), doi:10.1016/j.websem.2011.08.002

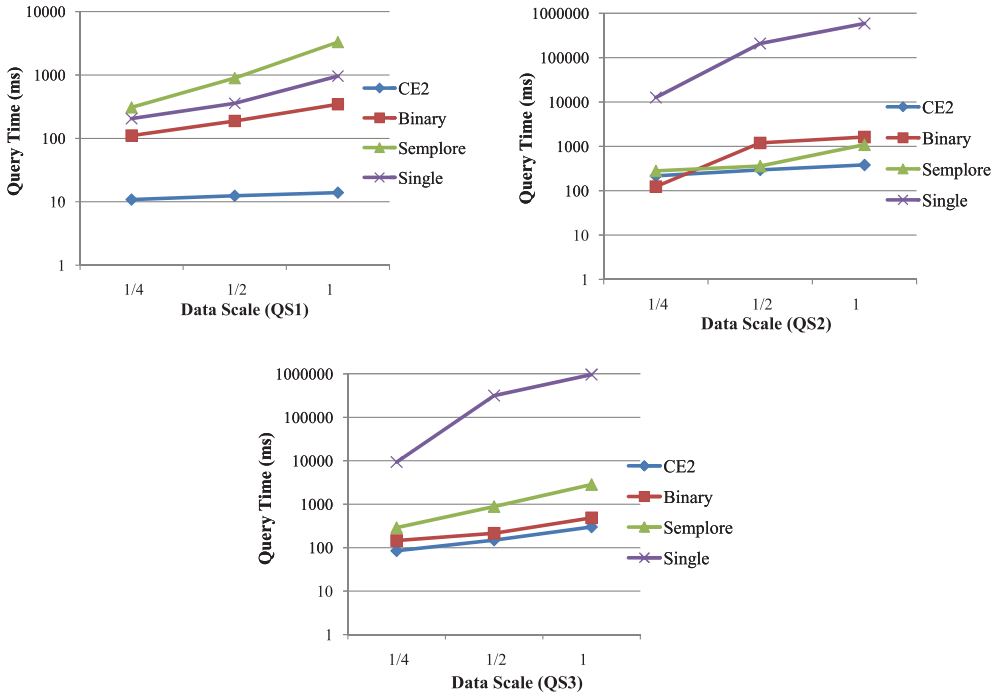

Fig. 7. Average response time (ms) for QS1, QS2 and QS3.

concepts extracted from documents. While these approaches focus on document retrieval, the models proposed in [37,22,6] are used for both Web documents and structured data. In particular, both structured data (in the form of annotations) and text in documents are used to match a given query, and merged to produce and rank the final answers [6]. Our previous work [35] generalizes and combines the above ideas to arrive at a framework and the main components of hybrid search. The work we present here provide additional details on this theoretical search framework. Furthermore, it contains a more elaborated discussion of the technical and implementation aspects and in particular, provides concrete techniques for efficient data storage and hybrid query processing.

Storage of RDF data – the design employed for data storage is closely related to approaches that adopt IR technologies to handle RDF data. Most notably, [34] proposes the use of the inverted index for RDF storage. This idea has been implemented in Semplore, which is used as a baseline system in our experiments. In [19], a sparse index has been exploited for the distributed processing of queries on RDF data. We have adopted the concepts behind these approaches to handle the large amount of graph structured data, annotations and texts in CE^2. In particular, CE^2 is based on the schema design discussed in [1,12,26]. More precisely, the database schema used in CE^2 is similar to the one employed for SOR [26]. The way annotations are managed in CE^2 is similar to how triples are indexed and stored in Semplore. However, while these approaches target a specific type of data, our goal is to manage all of them in an integrated fashion. For this, we elaborate on an additional layer that employs specific data structures and algorithms to combine data and ranking scores from different repositories and ultimately, to integrate inverted indexes with databases.

IR and DB integration – many commercial databases feature in-built indexes that can deal with textual data. Hybrid query processing is possible in such databases because they also allow for matching keywords against textual attributes [10]. There are more sophisticated systems for hybrid search that integrate more advanced IR and DB functionalities. The QUIQ engine [23] introduces a special hybrid query model based on two constraints: the *match* constraint corresponds to our `keyword` predicate, and the *filter* constraints correspond to the relation or attribute predicates of our query model. In order to computes candidates using both structured and textual data, QUIQ combines these different types of data in a single index where structured data is treated as "pseudo-keywords". The TopX engine [32] supports top-*k* retrieval on textual and semi-structured (XML) data. It supports queries that might contain content-specific constraints and content-and-structure constraints. Scores obtained for query parts are aggregated to obtain the total scores of final results that match the entire query. For this purpose, the proposed procedure for top-*k* processing and ranking can incorporate any scoring function that is monotonic. While these systems specifically deal with tree-structured XML data, the work we proposed applies to general graph-structured (RDF) data. For this, there is seminal work implemented for a system called CompleteSearch [5], which integrates DB functionality into an IR-based index that is customized to graph-structured RDF data. Recently, several systems [34,19] borrow similar idea for hybrid query processing over RDF data.

However, these approaches are focused on hybrid query processing on RDF, less on ranking which is a key aspect of this work. In fact, the kind of ranking that can be supported also makes up the main difference to the work on XML data such as TopX and QUIQ. The aggregation of scores supported by these approaches is similar to the mechanism for quality propagation proposed here. However, we do not only propagate scores along edges of the query but also, aggregate scores over the data elements that have been obtained for every single query edge. That is, the additional data structures and algorithms we propose allow to compute an aggregated score for results of every single query edge, and on top of that, to aggregate the scores of these intermediate results to obtain the score of the final query result. Worth mentioning also is that this work represents a lightweight integration of DB and IR, proposing the combination of off-the-shelf database with standard inverted index technology.

Ranking – different ranking schemes have been proposed for dealing with structured queries on (RDF) data. For instance,

Please cite this article in press as: H. Wang et al., Lightweight integration of IR and DB for scalable hybrid search with integrated ranking support, Web Semantics: Sci. Serv. Agents World Wide Web (2011), doi:10.1016/j.websem.2011.08.002

semantic search systems such as Sindice [30], Watson [15], Swoogle [16] and Falcons [14] provide lookup functionalities based on an IR engine such as Lucene. The Sindice system also applies ad-hoc rules such as "prefer data sources whose hostname corresponds to the resource's hostname". Ref. [4] computes scores using spreading activation, similarly to the ranking principles used in our approach, while [24] performs ranking according to factors such as extraction confidence and query length. Instead of a simple aggregation over term matching scores of predicate bindings like most DB and IR integration systems do (such as [32]), our ranking solution explicitly takes the graph structure into account for score propagation and aggregation. Beyond ranking schemes, we propose novel algorithms to tightly integrate such ranking scheme into hybrid query processing. Compared with ranking for relational databases (e.g., [7]), CE^2 computes the score for each cell w.r.t. a tuple and a given column or node in the query at runtime.

One issue we have not addressed in this paper is anti-spamming [27], we leave this as our future work.

Our previous proposal [35] introduced the main concepts of CE^2. Our current work gives further details and focuses on the technical and implementation aspects rather than on the modeling and formal definitions.

8. Conclusions and future work

We have elaborated on a model for hybrid search. With respect to this model, we have leveraged database and IR technologies to scale over large amounts of textual and structured data. In particular, we have presented algorithms and a data structure called OPT to support hybrid query processing against these resources. Ranking plays a central role in our hybrid search model and is thus tightly integrated into query processing. We have provided an implementation called CE^2 for data storage and hybrid query processing. While DB2 and Lucene have been employed as the underlying backend technologies for the experiments, CE^2 can be deployed on top of off-the-shelf databases and IR solutions. Our evaluation results are promising, showing that hybrid queries can be effectively answered using the proposed ranking scheme and efficiently processed in a large scale scenario.

Currently, only the IR matching score of keywords is used for ranking. We plan to consider further factors, both query-dependent and query-independent factors such as PageRank derived from the graph, and statistical dependencies of terms such as TF-IDF. An open question in this regard that we will investigate is how to combine scores derived from structural data and structural matching with scores derived from textual data and keyword-based matching. There is also potential for the adoption of more advanced query optimization strategies to achieve more efficient evaluation of hybrid queries. For this, we will study how existing optimization techniques can be extended to incorporate statistics for both structured query predicates and keyword query predicates. Another direction is the integration of top-k retrieval into hybrid query processing.

References

[1] D.J. Abadi, A.M. kala, S. Madden, K.J. Hollenbach, Scalable semantic web data management using vertical partitioning, in: C. Koch, J. Gehrke, M.N. Garofalakis, D. Srivastava, K. Aberer, A. Deshpande, D. Florescu, C.Y. Chan, V. Ganti, C.-C. Kanne, W. Klas, E.J. Neuhold (Eds.), VLDB, ACM, 2007.

[2] K. Aberer, K.-S. Choi, N.F. Noy, D. Allemang, K.-I. Lee, L.J.B. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, P. Cudré-Mauroux (Eds.), The Semantic Web, 6th International Semantic Web Conference, second Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11–15, 2007, Vol. 4825 of Lecture Notes in Computer Science, Springer, 2007.

[3] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, Z.G. Ives, Dbpedia: a nucleus for a web of open data, in: Aberer et al. [2].

[4] B. Bamba, S. Mukherjea, Utilizing resource importance for ranking semantic web query results, in: C. Bussler, V. Tannen, I. Fundulaki (Eds.), SWDB, vol. 3372, 2004.

[5] H. Bast, I. Weber, The completesearch engine: Interactive, efficient, and towards IR & DB integration, in: CIDR, 2007. Available from: <www.crdrdb.org>.

[6] R. Bhagdev, S. Chapman, F. Ciravegna, V. Lanfranchi, D. Petrelli, Hybrid search: effectively combining keywords and semantic searches, in: S. Bechhofer, M. Hauswirth, J. Hoffmann, M. Koubarakis (Eds.), ESWC, Vol. 5021 of Lecture Notes in Computer Science, Springer, 2008.

[7] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, S. Sudarshan, Keyword searching and browsing in databases using banks, in: ICDE, IEEE Computer Society, 2002.

[8] A. Bhaskar, C. Botev, M. Chettiar, L. Guo, J. Shanmugasundaram, F. Shao, F.Y. 0002, Quark: an efficient xquery full-text implementation, in: S. Chaudhuri, V. Hristidis, N. Polyzotis (Eds.), SIGMOD Conference, ACM, 2006.

[9] B. Bhattacharjee, S. Padmanabhan, T. Malkemus, M. Huras, Efficient query processing for multi-dimensionally clustered tables in db2, in: VLDB, 2003.

[10] C. Botev, J. Shanmugasundaram, S. Amer-Yahia, A texquery-based xml full-text search engine, in: G. Weikum, A.C. König, S. Deßloch (Eds.), SIGMOD Conference, ACM, 2004.

[11] S. Brin, L. Page, The anatomy of a large-scale hypertextual web search engine, Comput. Netw. 30 (1-7) (1998) 107–117.

[12] J. Broekstra, A. Kampman, F. van Harmelen, Sesame: a generic architecture for storing and querying rdf and rdf schema, in: Horrocks and Hendler [20], pp. 54–68.

[13] P. Castells, M. Fernández, D. Vallet, An adaptation of the vector-space model for ontology-based information retrieval, IEEE Trans. Knowl. Data Eng. 19 (2) (2007) 261–272.

[14] G. Cheng, Y. Qu, Searching linked objects with falcons: approach, implementation and evaluation, Int. J. Semant. Web Inf. Syst. 5 (3) (2009) 49–70.

[15] M. d'Aquin, M. Sabou, M. Dzbor, C. Baldassarre, L. Gridinoc, S. Angeletou, E. Motta, WATSON: A gateway for the semantic web, in: Poster Session at ESWC, 2007.

[16] L. Ding, T. Finin, A. Joshi, R. Pan, R. Cost, Y. Peng, P. Reddivari, V. Doshi, J. Sachs, Swoogle: a search and metadata engine for the semantic web, in: Proceedings of the 13th ACM International Conference on Information and Knowledge Management, ACM, 2004.

[17] O. Erling, I. Mikhailov, RDF support in the virtuoso DBMS, in: S. Auer, C. Bizer, C. Müller, A.V. Zhdanova (Eds.), CSSW, Vol. 113 of LNI, GI, 2007.

[18] I. Frommholz, N. Fuhr, in: G. Marchionini, ML. Nelson, CC. Marshall (Eds.), Probabilistic object-oriented logics for annotation-based retrieval in digital libraries, JCDL, ACM, 2006.

[19] A. Harth, J. Umbrich, A. Hogan, S. Decker, Yars2: a federated repository for querying graph structured data from the web, in: Aberer et al. [2].

[20] I. Horrocks, J.A. Hendler (Eds.), The Semantic Web – ISWC 2002, First International Semantic Web Conference, Sardinia, Italy, June 9–12, 2002, in: Proceedings, Vol. 2342 of Lecture Notes in Computer Science, Springer, 2002.

[21] I. Horrocks, S. Tessaris, Querying the semantic web: a formal approach, in: Horrocks and Hendler [20]. Available from: <http://link.springer.de/link/service/series/0558/bibs/2342/23420177.htm>.

[22] T. Immaneni, K. Thirunarayan, A unified approach to retrieving web documents and semantic web data, in: E. Franconi, M. Kifer, W. May (Eds.), ESWC, Vol. 4519 of Lecture Notes in Computer Science, Springer, 2007.

[23] N. Kabra, R. Ramakrishnan, V. Ercegovac, The quiq engine: a hybrid IR DB system, in: U. Dayal, K. Ramamritham, T.M. Vijayaraman (Eds.), ICDE, IEEE Computer Society, 2003.

[24] G. Kasneci, F.M. Suchanek, G. Ifrim, M. Ramanath, G. Weikum, NAGA: searching and ranking knowledge, in: ICDE, IEEE, 2008.

[25] E.P. Klement, R. Mesiar, E. Pap, Triangular Norms, Kluwer, Dordrecht, 2000.

[26] L. Ma, C. Wang, J. Lu, F. Cao, Y. Pan, Y. Yu, Effective and efficient semantic web data management over DB2, in: J.T.-L. Wang (Ed.), SIGMOD, ACM, 2008.

[27] P.T. Metaxas, J. DeStefano, Web spam, propaganda and trust, in: AIRWeb, 2005.

[28] P. Mika, E. Meij, H. Zaragoza, Investigating the semantic gap through query log analysis, in: A. Bernstein, D.R. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, K. Thirunarayan (Eds.), International Semantic Web Conference, Vol. 5823 of Lecture Notes in Computer Science, Springer, 2009.

[29] C. Murray, Oracle Spatial Resource Description Framework (RDF), 10g Release 2 (10.2), 2005.

[30] E. Oren, R. Delbru, M. Catasta, R. Cyganiak, H. Stenzhorn, G. Tummarello, Sindice.com: a document-oriented lookup index for open linked data, Int. J. Metadata Semantics Ontol. 3 (1) (2008) 37–52.

[31] T. Roelleke, J. Wang, TF-IDF uncovered: a study of theories and probabilities, in: S.-H. Myaeng, D.W. Oard, F. Sebastiani, T.-S. Chua, M.-K. Leong (Eds.), SIGIR, ACM, 2008.

[32] M. Theobald, H. Bast, D. Majumdar, R. Schenkel, G. Weikum, Topx: efficient and versatile top-k query processing for semistructured data, VLDB J. 17 (1) (2008) 81–115.

[33] D.T. Tran, S. Bloehdorn, P. Cimiano, P. Haase, Expressive resource descriptions for ontology-based information retrieval, in: ICTIR'07, 2007. Available from: <http://www.aifb.uni-karlsruhe.de/WBS/pha/publications/ontology-ir-ictir07.pdf>.

Please cite this article in press as: H. Wang et al., Lightweight integration of IR and DB for scalable hybrid search with integrated ranking support, Web Semantics: Sci. Serv. Agents World Wide Web (2011), doi:10.1016/j.websem.2011.08.002

[34] H. Wang, Q. Liu, T. Penin, L. Fu, L. Zhang, T. Tran, Y. Yu, Y. Pan, Semplore: a scalable IR approach to search the web of data, J. Web Sem. 7 (3) (2009) 177–188.
[35] H. Wang, T. Tran, C. Liu, Ce2: towards a large scale hybrid search engine with integrated ranking support, in: J.G. Shanahan, S. Amer-Yahia, I. Manolescu, Y. Zhang, D.A. Evans, A. Kolcz, K.-S. Choi, A. Chowdhury (Eds.), CIKM, ACM, 2008.
[36] K. Wilkinson, C. Sayers, H. Kuno, D. Reynolds, Efficient RDF Storage and Retrieval in Jena2, in: SWDB, vol. 3, 2003.
[37] L. Zhang, Y. Yu, J. Zhou, C. Lin, Y. Yang, An enhanced model for searching in semantic portals, in: A. Ellis, T. Hagino (Eds.), WWW, ACM, 2005.

Please cite this article in press as: H. Wang et al., Lightweight integration of IR and DB for scalable hybrid search with integrated ranking support, Web Semantics: Sci. Serv. Agents World Wide Web (2011), doi:10.1016/j.websem.2011.08.002