Enhancing Search with Structure

Soumen Chakrabarti Sunita Sarawagi S. Sudarshan

Computer Science and Engineering Department
Indian Institute Of Technology, Bombay, India
{soumen, sunita, sudarsha}@cse.iitb.ac.in

Abstract

Keyword search has traditionally focussed on retrieving documents in ranked order, given simple keyword queries. Similarly, work on keyword queries on structured data has focussed on retrieving closely connected pieces of data that together contain given query keywords. In recent years, there has been a good deal of work that attempts to go beyond the above paradigms, to improve search experience on unstructured textual data as well as on structured or semi-structured data. In this paper, we survey recent work on adding structure to keyword search, which can be categorized on three axes: (a) adding structure to unstructured data, (b) adding structure to answers, and (c) adding structure to queries allowing more power than simple keyword queries, but while avoiding the complexity of elaborate query languages that demand extensive schema knowledge.

1 Introduction

Web search and information retrieval (IR) have traditionally focused on retrieving documents in ranked order, given simple keyword queries. Meanwhile, work on keyword queries on structured data has focused on retrieving closely connected pieces of data that together contain given query keywords. Recent years have witnessed many attempts to go beyond the above paradigms, to improve search experience on data sources (see Section 2) ranging from relational data with textual fields to semi-structured graphs with text associated with nodes and edges to completely unstructured text. In this paper, we survey recent efforts toward adding structure to keyword search.

In Section 3 we review a few data models somewhere in the gap between relational tables from the database community and the sequence-of-tokens or vector-space models from the IR community. We are specifically interested in lowest common denominator representations of ER graphs, text segments connected to type hierarchies with attached entity nodes, RDF, and tables represented using HTML markup without formal schema.

Then, in Section 4.1, we discuss various possible definitions of the *unit of answer*, which is a tuple in relational queries and a document in IR. In the ER graph model, an answer may be a single node, or a small tree or subgraph. The answer may be an entity node in a type hierarchy. The answer may be a table, even if the source is unstructured text. The (initial) answer may be a form that the user has to fill to issue a more precise second query. In Section 4.2, we move on to the design of structure-enabling features in queries, while avoiding the complexity of full query languages.

Copyright 2010 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

In Section 5 we outline algorithmic and systems issues in efficiently evaluating some forms of structured queries, as well as queries that provide some forms of structured answers. We first summarize how to annotate unstructured text with links to entities in a graph-structured entity and type catalog. Then we discuss indexing and query processing for entity search in annotated text. Next we describe how to respond to queries with subgraphs, forms, and tables.

2 Structure in data sources

Databases and text search systems evolved very differently for several decades, but new applications are pushing them toward common ground [15]. In this section, we describe three types of data sources useful in new applications somewhere between strictly structured and completely unstructured data management systems.

We begin in Section 2.1 with the most explicit form of structure, where the job is to index and search entity-relation (ER) tables or graphs, except that there may be free-form text associated with table cells or nodes and edges in the graph. Standard relational or XML search systems do not provide sufficiently flexible support for combining text and structure search in such settings.

We continue in Section 2.2 with a less friendly situation where record-like regular structure has to be automatically extracted from HTML source. Here HTML markup tags and their regular patterns provide clues to extracting structure.

The most challenging scenario is where there is no explicit structure in the source text, but the structure is created by linking text tokens to entities in a structured database, which might be tabular or graphical. This is discussed in Section 2.3.

2.1 Structured data with textual content

Relational databases are widely used to store and generate textual content for e-commerce and social media sites. Familiar examples in academia include DBLP and CiteSeer. Many table columns in such applications are free-form or somewhat stylized text. The dominant query paradigm combines small forms for structured (e.g., numeric) fields and text boxes for free-format text input. Unlike with SQL queries where any row ordering must be deliberate and explicit, users expect tuples to be implicitly sorted by a relatively vague notion of relevance, as in IR. Finally, data normalization and the use of primary and foreign keys across multiple tables means that different words in a query may match cells in different tables, connected via joined attributes. We can think of every row in every table as a node in a graph, with edges representing primary and foreign key matches.

This "lowest common denominator" graph representation is quite generic, and can also be used to represent XML documents with text "blobs" associated with leaf nodes, and nodes being labeled with some element type. This kind of labeled entity-relation graph can also be used to represent RDF data. A specific and interesting example for us is DBPedia [1], which is an RDF repository built by extracting triples from Wikipedia.

2.2 Records and tables embedded in HTML

Web pages often contain embedded tables and lists that can be interpreted as sets of multi-attribute records. Such sources can serve as a compact and high quality source of structured information, that has been ignored until recently [9,11,21,23]. For example, researchers at Google report the extraction of 154 million information bearing tables from their search engine crawl [11]. In addition to providing partially structured sets of records, such artifacts also provide useful meta-data in the form of headers and titles. With proper tools, it is possible to collect several useful kinds of information such as instances of type-entity pairs from a table's header and its entity columns, and multi-way relationships from rows of a table. Specifically under the assumption that the first

[1] `http://dbpedia.org/About`

non-numeric unique column is an entity, tables can be used to extract (entity, attribute name, value) triples [10]. Lists on the web can be converted to yield useful multi-way relationships as discussed in [23]. The headers of tables provide an implicit grouping of attribute names that can be exploited in useful ways. For example, by exploiting the co-occurrence pattern of a column header with headers of other tables, one can group attribute names that are synonyms [11].

2.3 Unstructured text mentioning entities from catalog

The most "organic" sources of structure are in unstructured text on Web pages, and these are the most difficult to exploit reliably in search. In general, an *annotation* associates a short token segment with either a *type* of entities, or a specific entity. Named entity taggers can associate a coarse-grained type like *person* with the token sequence *Albert Einstein* with high accuracy. More fine-grained annotations may assert that tokens *Albert Einstein*, in a given context, refers to a *physicist* rather than a more generic type *person*. The most unambiguous level of information is that these tokens refer to a known entity, possibly expressed by a standard URN in an entity *catalog* such as Wikipedia.

Unstructured text in Wikipedia articles already have references to standard entity URNs within Wikipedia, and there are also many links from other pages that refer to a specific entity, to the corresponding URN in Wikipedia. Such annotated text leads to a new document representation (Section 3.2) that can be used for entity search. Authoring entity-annotated text is labor-intensive. Much of the Web is not annotated thus. In Section 5.1.2 we will summarize recent algorithms to automatically annotate free-form text with likely mentions of entities from a given catalog. These automatic annotation algorithms are not perfect, but the hope is that they have sufficient accuracy to assist semi-structured search.

3 Data representation

In Section 2 we discussed various forms in which structure may be explicit in, or be mined out of a data source. In this section we describe the corresponding abstract representations used in building storage and query systems. These representations bring together tabular relational data, document representation in traditional IR, and graph-structured data.

3.1 Entity and type catalog

A catalog provides a directed acyclic graph where nodes are *types* and *entities*. A directed edge labeled "$\subseteq$" from type T_1 to T_2 means that T_2 is a subtype or subclass of T_1. E.g., *Physicist* is a subtype of *Scientist*. Entities e are also represented as nodes, connected to types T they belong to, using edges labeled "$\in$". The catalog may be provided by WordNet [34], Wikipedia, or YAGO [43].

Each entity (or type) in the catalog is known by a unique ID. It is also known by one or more *lemmas*: short token sequences describing the entity (or type). E.g., the city of New York is known by lemmas "New York City" and "Big Apple". Over and above lemmas, each entity also has an associated *description* that uniquely defines it. In case of WordNet, the description is a dictionary definition. In case of Wikipedia, the description is a whole article page. More generally, we may find hyperlinks anywhere on the Web (including Wikipedia itself) to an entity e, and consider the anchor and surrounding text as another form of "description" of e.

3.2 Text linked to catalog

The vector space model [44] has served its purpose well in traditional IR, but more elaborate models are needed if we are to augment the query system with more structure. One way to do so is to exploit connections between the text and a catalog of types and entities. An *annotation* is a record with fields: document ID, range of token offsets, entity (or type) ID from the catalog, and an optional score of confidence

We might represent the corpus in graph format as a set of chains, one for each document. Each node in a chain represents a token in the corresponding document. In case a token is part of a suspected *mention* of entity e, the node corresponding to the token is attached by an edge to the node corresponding to e. The weight of this edge may be designed to reflect the (un)certainty of the mention. In case of multi-token mentions, depending on the application, one may add edges to each token or to the first, center or last token, with weights adjusted suitably.

In some cases, as in explicit hyperlinks from Web or Wikipedia documents to e, there is no uncertainty. But a majority of potential mentions are uncertain. E.g., the token segment "New York" may refer to either the city or the state, and "Einstein" may refer to the famous physicist or the well-known bagel makers. Many pages in Wikipedia are *disambiguation* pages that list alternative entity references for a given lemma.

3.3 General graphs with textual properties

In the above model of text linked to a catalog, the graph is very specific (catalog graph connected to document token chains) and there are only a handful of relationships: entity is instance of type, type is subtype of type, token/s mention entity, and tokens are adjacent/nearby. In a more general entity-relation (ER) graph model, nodes are arbitrary entities of given types. E.g., in representing DBLP, a node may represent a paper or author. Entity nodes may be explicitly connected by edges to a type "meta data" node, or the type may be interpreted as a discrete attribute of the entity node. In addition, nodes may have other named attributes; these could be numeric (e.g. date of publication) or string-valued. The string attributes may be short (such as author address) or long (an abstract).

Edges represent arbitrary binary relations between the nodes they connect. Like nodes, edges have types. E.g., a node representing a person may be connected to a node representing a paper, using an "is author of" edge. Generally, most binary relations have transposed relations, and we can model this as a reversed parallel edge, say, "is authored by" in the above example.

We can tokenize (long) text fields and represent each token as a node in the same ER graph. Each token node would then be connected to the entity nodes where they appear in attribute fields. We might define a "token appears in node" edge type to represent these relations.

3.4 RDF

The Resource Description Framework (RDF) is a framework originally designed for representing information on the Web. It is based on a set of (subject, predicate, object) triples, where subject and object are nodes of a graph, and the predicate (or property) is the label of an edge from subject to object. Each node may represent an entity, in which case it has a unique identifier, or it may represent a literal such as a string or number. Attribute values relationships are represented using edges. For example, we may have a graph with entity nodes "Jim Gray" and "Transaction Processing Concepts", the book written by Jim Gray. The Jim Gray entity may have an edge HasName linking it to a node representing the string "Jim Gray". and similarly the book may have an edge HasTitle linking it to a node representing the string "Transaction Processing Concepts". Although other graph models have been proposed for semistructured data, such as Lore [?], RDF is today the most widely used semistructured data model.

An RDF graph is conventionally represented as a set of triples, where each triple asserts a fact; an RDF graph asserts the conjunction of all triples in the graph. Unlike full featured knowledge representation languages, RDF does not allow specification of disjunction, negation, or quantification. In this aspect RDF is like the relational model, but unlike the relational model RDF does not require a strict schema, and it is restricted to the triple representation. Although RDF was originally proposed as a way for Web sites to provide semantics for their content, it has not been widely used in that context, but is a convenient unified representation for knowledge extracted from a variety of sources. For example, the YAGO project [43] uses RDF to represent knowledge extracted from Wikipedia and WordNet, among other sources.

3.5 Tabular data

At the top-level, tabular data extracted from the Web represents sets of records where the record boundaries are indicated by rows for tables and items for lists. Records comprise of a set of attributes. For lists, the attribute structure is latent and expensive extraction methods are required to reveal it (Section 5.4). For tables, the columns often correspond to attributes. Further, these can be partitioned relatively easily into those that represent entities, and those that are properties or attributes of the entities. Tables can optionally contain one or more header rows, and titles/context tokens describing the content.

Normally, the cells and headers of a table are not tied to any standardized schema. Greater leverage is possible by inter-linking the information across the millions of tables on the Web. One way to acheive this is to infer join links among table pairs — for example, Cafarella et al [11] propose to add join links between a column C of table T to a column C' of another table T' when the headers of C and C' are synonyms and co-occur with similar columns. A second approach is to annotate table columns and entities to a standardized catalog. In Section 3.2 we discussed the annotated text model where text snippets are annotated with entity nodes of a type hierarchy. For tables, in addition to entity annotation on the cells, type annotations can be attached to columns and relationship annotations among groups of columns. For example, based on its contents, a table's columns can be assigned type annotation such as "Western Movies", "Currency", "American director" and associated with relationship tags such as "box office total" and "directed by".

4 Models for queries and answers

To take advantage of explicit structure in the data, or latent structure in a corpus, we need one or both of the following features in the system:

- The unit of an *answer* has to be defined carefully. Unlike in Web search or IR, the answer will usually not be a document or page. As we shall see in Section 4.1, an answer to a keyword query may be a node in a graph, or a small subgraph, a table with textual and other fields, or even a form to fill for a second round of more structured query.
- There must be constructs in the query language to express more structure than in traditional keyword search systems. However, in doing so, the query language should not demand complete schema knowledge from users and border on to SQL or XQuery. Striking a balance in this matter is an important design issue. We will visit some promising query dialects in Section 4.2.

4.1 Answer structure for keyword queries

The unit of an answer to a keyword query on structured data can be defined in several ways, including nodes, trees, and graphs. Alternatively a keyword query can retrieve not raw data, but forms, or form results. We outline these options in this section.

4.1.1 Single node in a graph as answer

A common search task in the ER graph model is to rank single nodes in response to keyword queries. In standard IR, the score of a document in response to a query would be a function of the overlap of query words with text in the document. In the graph ER model, however, we would be interested in taking advantage of the graph structure for better ranking of response nodes.

As a specific motivating example, consider the DBLP or CiteSeer graphs. The paper about the Object Exchange Model (OEM) [38] is widely cited by papers about XML search, but it was written before XML became a mainstream research area in databases. Given that the OEM paper is directly related to XML technology and highly cited, one may argue that it should be ranked highly in response to a query containing 'XML'. As a more generic motivation, an edge between nodes u, v is evidence of an association, which might lead us to postulate that the score of of node u in response to a query should not be too different from the score of node v.

Proximity search approaches, such as [22], the near query mechanism in BANKS [1], and OBJECTRANK [7], return an entity node as an answer to a keyword query. For example, in the BANKS system, a query "author(near recovery)" would return authors that are closely related to the keyword recovery. The key issue is how to rank nodes in the input graph, in response to the given keyword query. Goldman et al. [22] find shortest paths from each potential answer node to each node containing the given keywords, and aggregate the shortest path lengths to get a node score. In constrast, BANKS [1] is based on a spreading activation model.

4.1.2 Trees and graphs as answers

A majority of the work on keyword search on structured data is based on finding a subtree connecting the keywords specified in the query. Early examples of this approach include BANKS [8], DBXplorer [4] and Discover [24].

While some early work ranked trees based on the number of edges, BANKS [8] proposed a more sophisticated ranking model based on edge directionality and edge weights; this model was later refined based on traversal probabilities in [26]. The primary motivation for the more sophisticated ranking approach was that unrelated nodes often have very short parts connecting them through "hub" nodes; for example if a paper node links to the conference in which it appears, every two papers in that conference have a path of length 2 connecting them, even if they are otherwise unrelated. The directed edge weight model penalizes such shortcut paths that connect a large number of nodes.

When nodes can contain significant amounts of text, IR ranking techniques are important for computing answer tree scores. Fang et al. [32] describe how to apply IR ranking techniques such as TF-IDF, document size normalization and phrase-based ranking to rank answer trees.

Given a keyword query, Précis [28] first finds nodes containing the keywords; but instead of just returning these nodes, for each such answer node Précis returns a subgraph of related nodes. The subgraph of related nodes is generated based on foreign key joins with other closely related relations, which can be automatically chosen based on some heuristics.

The Précis approach starts with a node or a path/tree and grows a graph around it. An alternative approach is based on finding small subgraphs that contain the given query keywords. For example, the EASE system [31] finds r-radius Steiner subgraphs that contain the given query keywords, ranked based on their compactness and length-normalized TF-IDF scores.

4.1.3 Forms and Queries as Answers

Presenting nodes, trees or graphs as answers is often not satisfactory for end users who are not concerned with the schema. For example, users of enterprise information systems are used to getting richly structured answers in response to filling in forms. A different approach is required to generate such structured answers; in particular, answers formats need to be predecided in some way, and a keyword query retrieves one of the predecided answer formats.

One approach, proposed by [17], precomputes a set of forms, with associated SQL queries, from the given database schema. A keyword search then returns a set of forms that are relevant to the query, instead of returning actual data; a form is considered relevant to a query if the keywords occur in relations used in the form query. The user can then provide values for form parameters to get at required information.

A second approach, proposed by [36], is to specify units of information called qunits, each defined using a parametrized query, defined in a language such as SQL/XML. Executing each such qunit template with different instantiations of the parameters leads to a collection of instantiated qunits. Keyword queries are evaluated against the set of instantiated qunits, and relevant qunits are returned as (ranked) answers. A major focus of [36] is on how to automate the generation of qunits, and a user study shows that automated generation can perform reasonably close to human generated qunits.

A third approach is applicable to a situation where an enterprise information system has already defined a large number of forms; here, the goal of a keyword query is to find a form, along with an instantiation of form parameters, such that the result of the instantiated form contains the specified keywords. In the approach proposed by [20], forms results are precomputed for all possible parameter values, and indexed offline. In enterprise information systems, this approach is complicated by the fact that different users may see different results on submitting the same form. In Section 5.3 we outline how Duda et al. [20] handle this problem. The problem of deep Web search, e.g. [40], is closely related. Work in this area has generally focussed on generating values to be filled into Web forms to surface data hidden behind the forms; pages thus surfaced are then indexed just like regular Web pages.

4.1.4 Tables as Answers

For many user information needs, the answer is best represented as a single table rather than as a collection of documents. For example, in Google Squared [2] a search for entity types such as “fighter jets” outputs a table with rows representing instances of fighter jets and columns denoting attributes like dimensions and crew size. The column of the table are either specified by the user or inferred by referring to an offline extracted knowledge base of attributes and classes [39]. Typically, such tables are constructed from noisy consolidation of information from multiple sources. So, the cells of the table contain multiple ranked plausible answers.

4.2 Query structure

Several approaches have been proposed for adding some degree of structure to keyword queries. We outline a few of these in this section.

4.2.1 Type and proximity queries

A broad class of *entity search* queries ask about an unknown entity belonging to a known type that satisfy some properties. These properties are loosely expressed in terms of the entity being mentioned in close juxtaposition with certain keywords. E.g., using the category *Physicist* in Wikipedia, we might look for physicists who play(ed) the violin by searching for short token sequences that contain both a descendant of type *Physicist* and the literal word *violin*. In Section 5.1.3 we present various issues of answering such queries. Searching for entities is now a standard track in the TREC and INEX competitions.

4.2.2 EntityRank

The next step is to ask for not one entity of a given type but a few entities with given types, related to each other in specified ways [16]. (The relation is again favored in a soft manner through lexical proximity within a short token sequence.) E.g., if we had surface recognizers (Section 2.3) for emails and phone numbers, and we managed to reliably annotate entities that are instances of *database researcher*, we could ask for a table with three columns: the mention of a database researcher (“David Lomet”), an email (`lomet@xyz.com`) and a phone number (“800-555-1212”). Note that the use of schema in the query “language” thus far is not very

[2] `http://www.google.com/squared`

overbearing. The only new query construct is a *type name*. Of course, the user has to know the type name from a standard catalog, but autocompletion and query suggestions can help [42].

4.2.3 Adding variables, contexts, and joins

Pushing further on the paradigm of structured queries on unstructured data, the logical next stop involves *variables* and *joins*. Suppose we can name a variable `?m` as a placeholder for a *movie*, variable `?o` as a number, and variable `?b` as a money amount, expressed as these subqueries:

- $\texttt{?m} \in^+ \textit{Type:Movie}$
- $\texttt{?o} \in \textit{Quantity:Number}$
- $\texttt{?b} \in^+ \textit{Quantity:MoneyAmount}$

(Here "$\in^+$" denotes transitive membership via intermediate subtypes.) Then we can ask for token segments or *contexts* `?c1` and `?c2` such that

- $\textsc{InContext}(\texttt{?c1}; \texttt{?m}, \texttt{?o}; \textit{won}, \textit{Oscar})$
- $\textsc{InContext}(\texttt{?c2}; \texttt{?m}, \texttt{?b}; \textit{production}, \textit{cost}, \textit{budget})$

and finally aggregate over contexts `?c1` and `?c2` to get a table with three columns: a movie name mention, the number of Oscars it won, and its production budget. Although such a language may not be used by end-users, it can be used as a "decision support" tool.

4.2.4 Specifying graph structures

Keyword queries can specify partial structures on graphs. For example, XPath expressions, or approximate match XPath expressions can be used to restrict the scope of a keyword query on XML data; see for example, [6], which explores flexible path specifications in the context of XML data. The idea of flexible path specifications can be easily applied to graph data. In the context of RDF graphs, one could use SPARQL, a structured, SQL-like query language for querying RDF.

The NAGA search engine [27] proposes a query language that allows specification of some kinds of graph constraints such as path and edge constraints. Unlike SPARQL, the graph-based NAGA query language takes into account uncertainty of underlying facts, and aggregates support for a fact from multiple sources. A comparison of result quality using NAGA with results of Google search, and results of keyword queries using the BANKS scoring model is presented in [27].

4.2.5 Specifying table structure

In Section 4.1.4 we discussed the Google Squared method of returning table answers to keyword queries. These provide limited control to the user on how to describe the answer table. We propose two alternative modes of querying for tables. The first kind of queries are called column description queries where the user fills in multiple keyword boxes where each text box describes a column of the answer table. For example to retrieve a list of scientists and their inventions, he fills two text boxes: one containing words like "inventor, scientist" and the second containing words like "major invention". The second kind are content queries where a user submits a few examples of structured rows of the desired table and expects to get more instances of such rows. These examples can prove invaluable for teaching an extractor how to convert unstructured list sources to structured tables in the answer [23].

5 Algorithms and systems

In this section we review some algorithmic and system issues that come up when one implements the annotator, index, and search capabilities introduced before.

5.1 Entity search in annotated text

5.1.1 Broad type annotation

Machine learning techniques have come a long way toward annotating text with coarse-grained types. Let a text token sequence be called $x = (x_1, x_2, \ldots, x_T)$, and designate (unknown) labels $y = (y_1, y_2, \ldots, y_T)$ to these token positions. Each y_t can take on type values like *Person*, *Street name*, *Paper title*, *Phone number*, and so on. Usually there is also a "none of the above" label for tokens not recognized to belong to any of the specified groups. Then the job is to find the best labeling y. Note that, if each token position can have one of M labels, the number of possible labelings is astronomical, M^T. This explosion can be avoided using the Viterbi algorithm used in hidden Markov models. Recent techniques have improved upon HMMs by defining general families of *feature functions* $\phi(x, y) \in \mathbb{R}^d$ and then learning a *model* $w \in \mathbb{R}^d$ such that the best labeling is given by $\arg\max_y w^\top \phi(x, y)$ [41].

5.1.2 Entity disambiguation

Beyond broad types, we want to annotate tokens with disambiguated entities. Suppose s is a spot (token segment) in a document that is suspected to mention some entity from a set Γ_s. It is possible that s does not mention *any* entity in Γ_s, and the "null" label is denoted $N.A.$. Let S_0 be the set of all spots in a document. Then the job is to pick, for each s, an entity label $\gamma \in \Gamma_s \cup \{N.A.\}$. Recall that γ comes from an entity catalog. Specifically, if WordNet is the catalog, each concept has a dictionary-like definition with an example "gloss" showing typical usage. If the catalog is Wikipedia, there is a whole article describing the entity at length. On the other side, the potential mention s has a textual context. We would generally expect some affinity between the context of s and the definition of γ. This can be expressed in a manner neutral to the specific identity of s or γ, as a feature vector $f_s(\gamma)$ in some space. Again, we can learn a model w in the same space such that the predicted entity label is $\arg\max_{\gamma \in \Gamma_s \cup N.A.} w^\top f_s(\gamma)$ [29,33].

However, one may do better by exploiting the fact that entities mentioned in a single discourse tend to be semantically related. E.g., Wikipedia lists several Michael Jordans of who only one is a Computer Science professor, and likewise with Stuart Russell. However, a single Web page listing both names almost certainly refer to the two computer scientists. A collective optimization over all spot labels [19,35] can improve disambiguation accuracy noticeably.

5.1.3 Indexing and query processing

To process INCONTEXT queries as presented in Section 4.2.3, we need two basic devices:

- Find connections between types in the query to entities mentioned in token segments.
- Score and rank these token segments using textual proximity as an important feature.

Reachability testing between the query type and a candidate entity answer can be done in at least two ways:

- At query time, expand the query type into each candidate in turn, and score the candidates. In other words, if the query is looking for a *scientist* who studied whales, effectively ask if e and 'whale' appear in a context, where $e \in^+$ *scientist*. This slows down queries unacceptably: Even WordNet knows of 650 scientists and 860 cities, not to mention Wikipedia or YAGO.

- At indexing time, suppose a token is a mention of entity e, and let $\mathcal{T}_e$ be all types to which e belongs directly or transitively. Then we insert pseudotokens encoding all types $T \in \mathcal{T}_e$ at the same token position and index these. This works well in applications with a small type system (5–10) [16]. In contrast, Wikipedia and YAGO have over 250,000 types and over 2.2 million entities, and the average size of $\mathcal{T}_e$ may be 10–30. This leads to unacceptable index size blowup.

One approach [13] is to compromise on both ends: index a carefully chosen subset of types, then do more work at query time. I.e., follow a "pre-generalize/post-filter" paradigm. Given a query type a, if a has not been indexed, generalize to some type $g \supseteq^{+} a$, and launch an IR-style query. Afterward, check if the instance of g in each candidate context is an also instances of a; if not, discard. Experiments with TREC and WordNet suggest that index size can be kept comparable to a standard text index while slowing down queries by less than a factor of 2.

5.2 Ranking entity nodes in ER graphs

Proximity queries in ER graphs seek entity nodes that are "near" nodes representing query words, or entity nodes that match query words. This paradigm is easiest to express in terms of personalized PageRank [25, 37]. Each directed edge (u, v) of the data graph has a *conductance* $C(v, u) \in (0, 1)$, which is the conditional probability $\Pr(v|u)$ that a "random surfer", in PageRank parlance, will walk from node u in the previous step to node v in the current step, should he decide to walk, which happens with probability $\alpha \in (0, 1)$. By design, $\sum_v C(v, u) = 1$. With probability $1 - \alpha$, he *teleports* to a node w in the graph with probability $r(w)$, with $\sum_u r(u) = 1$. It is well-known that the PageRank vector p_r corresponding to teleport r satisfies the recurrence $p_r = \alpha C p_r + (1 - \alpha) r$, which solves to $p_r = (1 - \alpha)(\mathbb{I} - \alpha C)^{-1} r$. Summarizing, query processing consists of

- Designing the data graph, C and α ahead of time
- Using the query to define r
- Computing PageRank vector p_r
- Reporting the top K nodes u with the largest $p_r(u)$ scores.

A variant of the above paradigm (that allowed taking much of the computation offline) was proposed as OBJECTRANK [7]. The matrix C can also be learned automatically from training data [2, 3].

Traditional offline computation of PageRank is done by power iteration: Initialize p_r to some random nonzero vector, then iterate $p_r \leftarrow \alpha C p_r + (1 - \alpha) r$. For large data graphs (beyond tens of millions of nodes), it is not practical to compute p_r at query time using power iterations. The special structure of personalized PageRank can be exploited [12] to achieve an almost *constant* query time independent of graph size, provided special indices are precomputed whose size scales as a modest fraction of graph size.

5.3 Executing tree and form queries

In this section, we address the issue of answering tree and form queries, focusing on preprocessing/indexing, and query execution.

Work in the area of efficient computation of connection trees or graphs as answers to keyword queries can be divided into two approaches:

1. approaches based on generating and executing SQL queries, and
2. approaches based on graph traversal.

In a companion article in this issue, Yu et al. [45] provide an extensive survey of techniques for executing keyword queries on relational and graph databases.

Work on indexing deep Web content, such as by Raghavan and Garcia-Molina [40], is primarily based on surfacing the content by filling in form values, so that the content can be indexed subsequently. This approach is implemented in popular search engines today.

As an alternative, if the domain of an input to a form is known, e.g. a set of flight numbers, the set of values associated with the domain can be added to the document representing the form; a query containing these keywords could then retrieve the form.

In contrast to the Web scenario, in an enterprise setting the underlying data is available, and can be used to retrieve forms without surfacing all results of the form; however access restrictions must be taken into account. For example, in the approach of Duda et al. [20], each form is specified using a representation language which allows the system to understand what results are generated for what form parameters, with the identity of the user also treated as a parameter. Then, keywords from parts of the form that are dependent on a parameter value (for example, the name of a person where employee identifier is a parameter) are stored in a keyword index with an associated predicate.

For example, consider a form that outputs an employee name, given the employee identifier. If employee ID 2345 had the name John, the keyword index entry for John would include the above form, with an associated predicate “empID=2345”. A query “John” would retrieve an index entry with the above predicate, allowing the system to infer that the form, with parameter empId=2345 is an answer to the query. If the query contains multiple keywords, the inverted lists are merged, taking the predicates into account. Parts of the form which are common across all parameter values would not have any associated predicate, minimizing the storage and query cost.

5.4 Processing table queries

In this section we discuss how to process table search queries of the kind described in 4.2.5 by tapping tables and lists on the Web. We present a brief summary; more details can be found elsewhere [23][3].

5.4.1 Pre-processing and annotation

Recognizing useful tables and lists First, we need to identify parts of HTML pages that contain useful record sets. Depending only on HTML tags such as ⟨table⟩ and ⟨li⟩ is not sufficient as these are often used as layout tools or navigational aids for non-record data. For instance, as observed in [9] only 10% of HTML table tags are used to represent structured content. Similarly, only a small fraction of lists on HTML pages actually represent record sets.

Catalog annotations on web tables A catalog is used to annotate table cells with entity nodes, table columns with type nodes and column pairs with relationship identifiers. The lemma attached with these nodes are used to associate additional keywords with table cells and columns.

Indexing A text index is constructed on the extracted record sets where each set is associated with three kinds of fields: a bag of word representation of all the contents in the table, the catalog annotations, and the union of the title and descriptive context of the table.

5.4.2 Processing at query time

During query processing the index is first probed using the query words to find a list of candidate tables and lists. These are then subjected to the following steps:

[3] `http://www.cse.iitb.ac.in/~sunita/wwt`

Table extraction from HTML record sets There is a long history of extracting structure from record data on the Web [5, 14, 18, 21, 30, 46]. However, most of this work has been restricted to specific verticals like shopping, advertisements, and publications. Recent work [23] seeks robust extractors that do not depend on prior schema knowledge. At query time they have only (1) the few example rows provided by the user, and (2) other candidate record sets with overlapping content. The trick is to first use the query rows to create a partially labeled dataset, and then fit a model that simultaneously maximizes the probability of the labeled data and the probability that overlapping content across different lists get the same label.

Consolidation and ranking The extractor outputs a set of tables along with row and cell confidence scores indicating the probability of correctness of the extraction. The consolidator merges these tables into a single result table by resolving the set of rows that are duplicates. A cell in the consolidated table contains a set of cell values that are aliases of each other. Intuitively, a high scoring consolidated row is one which repeats in sources which overlap greatly with query words, has a high confidence score of extraction, and contains most of the useful columns.

6 Outlook

A great deal of recent work aims to improve the quality of results of keyword queries by adding structure to data, answers, and to queries themselves. We believe that the results of this research will have a significant impact on bridging the gap between structured and unstructured search. This article gives the reader a feel for the fast-growing area, but we cannot claim to have covered it exhaustively.

Acknowledgements: This work was supported in part by research grants from Microsoft and Yahoo.

References

[1] B. Aditya, S. Chakrabarti, R. Desai, A. Hulgeri, H. Karambelkar, R. Nasre, Parag, and S. Sudarshan. User interaction in the BANKS system (demo paper). In *ICDE*, pages 786–788, 2003.

[2] A. Agarwal and S. Chakrabarti. Learning random walks to rank nodes in graphs. In *ICML*, 2007.

[3] A. Agarwal, S. Chakrabarti, and S. Aggarwal. Learning to rank networked entities. In *SIGKDD Conference*, pages 14–23, 2006.

[4] S. Agrawal, S. Chaudhari, and G. Das. DBXplorer: A system for keyword-based search search over relational databases. *ICDE*, 2002.

[5] M. Álvarez, A. Pan, J. Raposo, F. Bellas, and F. Cacheda. Extracting lists of data records from semi-structured web pages. *Data Knowl. Eng.*, 64(2):491–509, 2008.

[6] S. Amer-Yahia, L. V. S. Lakshmanan, and S. Pandit. Flexpath: Flexible structure and full-text querying for xml. In *SIGMOD*, pages 83–94, 2004.

[7] A. Balmin, V. Hristidis, and Y. Papakonstantinou. ObjectRank: authority-based keyword search in databases. In *VLDB Conference*, 2004.

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

[9] M. Cafarella, N. Khoussainova, D. Wang, E. Wu, Y. Zhang, and A. Halevy. Uncovering the relational web. In *WebDB*, 2008.

[10] M. J. Cafarella. Extracting and querying a comprehensive web database. In *CIDR*, 2009.

[11] M. J. Cafarella, A. Halevy, Y. Zhang, D. Z. Wang, and E. Wu. Webtables: Exploring the power of tables on the web. In *VLDB*, 2008.

[12] S. Chakrabarti. Dynamic personalized PageRank in entity-relation graphs. In *WWW Conference*, Banff, May 2007.

[13] S. Chakrabarti, K. Puniyani, and S. Das. Optimizing scoring functions and indexes for proximity search in type-annotated corpora. In *WWW Conference*, Edinburgh, May 2006.

[14] C. Chang. and S. Lui. Iepad: Information extraction based on pattern discovery. In *WWW*, pages 681–688, 2001.

[15] S. Chaudhuri, R. Ramakrishnan, and G. Weikum. Integrating DB and IR technologies: What is the sound of one hand clapping? In *CIDR*, 2005.

[16] T. Cheng, X. Yan, and K. C. Chang. EntityRank: Searching entities directly and holistically. In *VLDB Conference*, pages 387–398, Sept. 2007.

[17] E. Chu, A. Baid, X. Chai, A. Doan, and J. Naughton. Combining keyword search and forms for ad hoc querying of databases. In *SIGMOD*, 2009.

[18] W. W. Cohen, M. Hurst, and L. S. Jensen. A flexible learning system for wrapping tables and lists in html documents. In *WWW*, 2002.

[19] S. Cucerzan. Large-scale named entity disambiguation based on Wikipedia data. In *EMNLP Conference*, pages 708–716, 2007.

[20] C. Duda, D. A. Graf, and D. Kossmann. Predicate-based indexing of enterprise web applications. In *CIDR*, 2007.

[21] H. Elmeleegy, J. Madhavan, and A. Halevy. Harvesting relational tables from lists on the web. In *VLDB*, 2009.

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

[23] R. Gupta and S. Sarawagi. Answering table augmentation queries from unstructured lists on the web. In *Proc. of the 35th Int'l Conference on Very Large Databases (VLDB)*, 2009.

[24] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-Style keyword search in relational databases. *VLDB*, 2002.

[25] G. Jeh and J. Widom. Scaling personalized web search. In *WWW Conference*, pages 271–279, 2003.

[26] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, pages 505–516, 2005.

[27] G. Kasneci, F. M. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum. NAGA: Searching and ranking knowledge. In *ICDE*, pages 953–962, 2008.

[28] G. Koutrika, A. Simitsis, and Y. E. Ioannidis. Précis: The essence of a query answer. In *ICDE*, pages 69–78, 2006.

[29] S. Kulkarni, A. Singh, G. Ramakrishnan, and S. Chakrabarti. Collective annotation of Wikipedia entities in Web text. In *SIGKDD Conference*, 2009.

[30] K. Lerman, C. Knoblock, and S. Minton. Automatic data extraction from lists and tables in web sources. In *Workshop on Advances in Text Extraction and Mining (ATEM)*, 2001.

[31] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *SIGMOD*, 2008.

[32] F. Liu, C. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *SIGMOD*, pages 563–574, 2006.

[33] R. Mihalcea and A. Csomai. Wikify!: linking documents to encyclopedic knowledge. In *CIKM*, pages 233–242, 2007.

[34] G. Miller, R. Beckwith, C. FellBaum, D. Gross, K. Miller, and R. Tengi. Five papers on WordNet. Princeton University, Aug. 1993.

[35] D. Milne and I. H. Witten. Learning to link with Wikipedia. In *CIKM*, 2008.

[36] A. Nandi and H. V. Jagadish. Qunits: queried units in database search. In *CIDR*, 2009.

[37] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the Web. Manuscript, Stanford University, 1998.

[38] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proc. of ICDE*, 1995.

[39] M. Pasca, B. V. Durme, and N. Garera. The role of documents vs. queries in extracting class attributes from text. In *CIKM*, pages 485–494, 2007.

[40] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *VLDB*, pages 129–138, 2001.

[41] S. Sarawagi. Information extraction. *FnT Databases*, 1(3), 2008.

[42] A. Singh, S. Kulkarni, S. Banerjee, G. Ramakrishnan, and S. Chakrabarti. Curating and searching the annotated web. In *SIGKDD Conference*, 2009. System demonstration.

[43] F. M. Suchanek, G. Kasneci, and G. Weikum. YAGO: A core of semantic knowledge unifying WordNet and Wikipedia. In *WWW Conference*. ACM Press, 2007.

[44] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan-Kaufmann, May 1999.

[45] J. X. Yu, L. Qin, and L. Chang. Keyword search in relational databases: A survey. *IEEE Data Eng. Bull.*, 33(1), Mar. 2010.

[46] Y. Zhai and B. Liu. Web data extraction based on partial tree alignment. In *WWW*, pages 76–85, 2005.