Towards Expressive Exploratory Search Over Entity-Relationship Data

Sivan Yogev
IBM Research Haifa, Israel
and
Department of Computer Science
Ben-Gurion University of the Negev, Israel
sivany@il.ibm.com

Haggai Roitman, David Carmel, Namma Zwerdling
IBM Research Haifa, Israel
{haggai,carmel,naamaz}@il.ibm.com

ABSTRACT

In this paper we describe a novel approach for exploratory search over rich entity-relationship data that utilizes a unique combination of expressive, yet intuitive, query language, faceted search, and graph navigation. We describe an extended faceted search solution which allows to index, search, and browse rich entity-relationship data. We report experimental results of an evaluation study, using a benchmark of several of entity-relationship datasets, demonstrating that our exploratory approach is both effective and efficient compared to other existing approaches.

Categories and Subject Descriptors: H.3.3 [Information Search and Retrieval]: Search process; Query formulation; Retrieval models

General Terms: Algorithms, Experimentation

Keywords: Entity-relationship data, Exploratory search

1. INTRODUCTION

Data complexity and its diversity have been rapidly expanding over the last years, spanning from large amounts of unstructured and semi-structured data to semantically rich available knowledge. Increasing demands for sophisticated discovery capabilities over rich *entity-relationship* (ER) data are now being imposed by numerous applications in various domains such as social-media, healthcare, telecommunication, e-commerce and web analytics, business intelligence, and cyber-security.

Many useful facts about entities (e.g. people, locations, organizations, products) and their relationships can be found in multitude semi-structured and structured data sources such as Wikipedia (*http://wikipedia.org*), Linked Data cloud (*http://linkeddata.org*), Freebase (*http://freebase.com*), and many others. Yet, many of these facts are hidden behind barriers of language constraints, data heterogeneity, ambiguity, and the lack of proper query interfaces. Therefore, novel discovery methods are required to provide highly expressive discovery capabilities over large amounts of entity-relationship data, which are yet intuitive for end-users.

ER discovery approaches can be classified according to two main *user-centric* aspects, namely the type of queries they support (termed *query type* hereinafter) and the amount of user involvement in the discovery process (termed *query execution* hereinafter).

Query types range from free-text queries to fully structured queries. Free-text queries allow end-users a simple way to express their information needs independently from the underlying data model and structure, as well as from a specific query language. On the other hand, structured query languages such as SQL for relational data, XQuery for XML, and SPARQL for RDF data, allow users to submit queries that may precisely identify their information needs, but often require users to be familiar with formal logic representation and with the underlying ontology and data structure.

Query execution ranges from *one-shot queries* to *iterative queries*. A one-shot query is executed once by the system without supporting additional user involvement. Therefore, the search system is solely responsible for satisfying the user's information needs. Inspired by interactive information retrieval [27], where end-users can interactively refine their queries whenever their initial information need is not satisfied, iteration-supporting systems allow a sequence of query refinements through user involvement during the iterative querying process.

We present a novel exploratory approach over ER data which combines the advantages of existing discovery approaches along the two aspects. Our approach is based on a combination of an *expressive query language*, *faceted search*, and *ER graph navigation*. Users can express their initial information need using a wide range of queries, spanning from simple free-text queries to more structured constraints over entity properties and their relationships with other entities. This allows utilization of a single search system to query entities and their relationships by both expert users and end-users who are neither familiar with the query language nor the data model.

We describe an extended faceted search solution for indexing and searching entity-relationship data. Our implementation generalizes the dual entity data representation previously proposed by Amitay et al. in the context of social search [2]. Each entity is dually represented as a searchable document and as a category of all other entities it relates to. Thus, the categories of retrieved entities enable browsing over the ER graph.

However, this dual representation supports only binary relationships. For example, a bookmark relationship in a social bookmarking system captures three entities involved in any bookmarking event: the tagged document, the tagger, and the tag. Such relationships are usually expressed using three different binary relationships [18]. In this paper we present a generalized solution which enables to capture arbitrary n-ary relationships by representing each relationship instance in the system as a *category set* that contains the categories of all participating entities. This representation is required when searching for specific relationship instances, for example, the collection of tags used by a specific person to bookmark a specific document – a set that cannot be identified by binary relations only. In Section 4 we expand on the implementation details of our system, and demonstrate how various rich entity-relationship data can be indexed and searched.

The output of the search system is a ranked list of entities that match the user query. Similarly to traditional faceted search systems, the system provides a distribution of retrieved entities over the facets they belong to, including facets of various entity types, properties, and related entities. The user in turn can exploit such facets to focus the search on a specific entity type or attribute, or to explore another direction by navigating to another related entity in the ER graph.

Whenever the user chooses to either filter the current result set based on some facet or to follow related entities using a relationship facet, the user query is automatically refined to reflect her choice using structured query predicates, while releasing the user from the need of understanding the underlying query language. On each iterative query step, similarly to traditional faceted search systems, the system provides the user with a useful report on facet distributions, revealing the number of sub-results expected for each facet choice.

Figure 1 demonstrates a discovery usage in our system for the social-medical domain [26] which is integrated within the IBM Patient Empowerment System[1]. In this example, two patients are returned as relevant to the initial query "*Hemophilia*" submitted by a user, who later on followed the "*Related patients*" link to discover related patients to the query. The system exposes several facets as well as other entities related to those patients. For each patient the user may further restrict the search to a specific facet, or follow relationship links to explore other related entities.

The main contributions of this paper can be summarized as follows.

• We describe a novel interactive querying approach over rich entity-relationship data which provides a unique combination of an expressive, yet intuitive, query language, faceted search, and ER graph navigation.

• We provide detailed description of a system which implements the aforementioned approach.

• Using experimental evaluation over several benchmarks of ER data, we demonstrate that our solution is both effective and efficient compared to other existing approaches.

2. RELATED DISCOVERY APPROACHES

Existing ER discovery approaches can be classified according to the *query type* and the *query execution* dimensions mentioned in Section 1. Next we discuss these classes in more details.

[1]IBM Patient Empowerment System social-medical discovery demo is available at http://www.youtube.com/watch?v=YFRjOB39hvA

One-shot free-text queries include simple keyword search services, and more recently, systems that allow users to submit keyword queries over structured data such as relational data [19, 1, 13], XML data [16], and RDF [23, 33]. Such approaches have become popular means for data discovery due to their simplicity and intuitiveness to end-users. However, interpreting user information needs expressed in free-text queries is usually not an easy task, especially due to the *vocabulary mismatch* between the user query and the data. In some cases, especially when structured data is queried, additional result analysis for the user query is needed (e.g., entity extraction), in order to correctly interpret user needs and translate them into the underlying data model.

One-shot structured queries eliminate some of the drawbacks of free-text queries by providing expressive means for query languages, e.g., SPARQL for RDF data [28]. Yet, such languages still have several limitations. First, users must be familiar with the query language syntax. Second, users may need to be familiar with the underlying data model and its semantics. The second limitation may be relaxed by using logical views over queried data [17]. However, such relaxation usually adds additional complexities to query execution such as the need for data mappings [17]. Therefore, systems that provide structured query interfaces have not been so far willingly adopted by end-users, but rather been mostly used by domain experts.

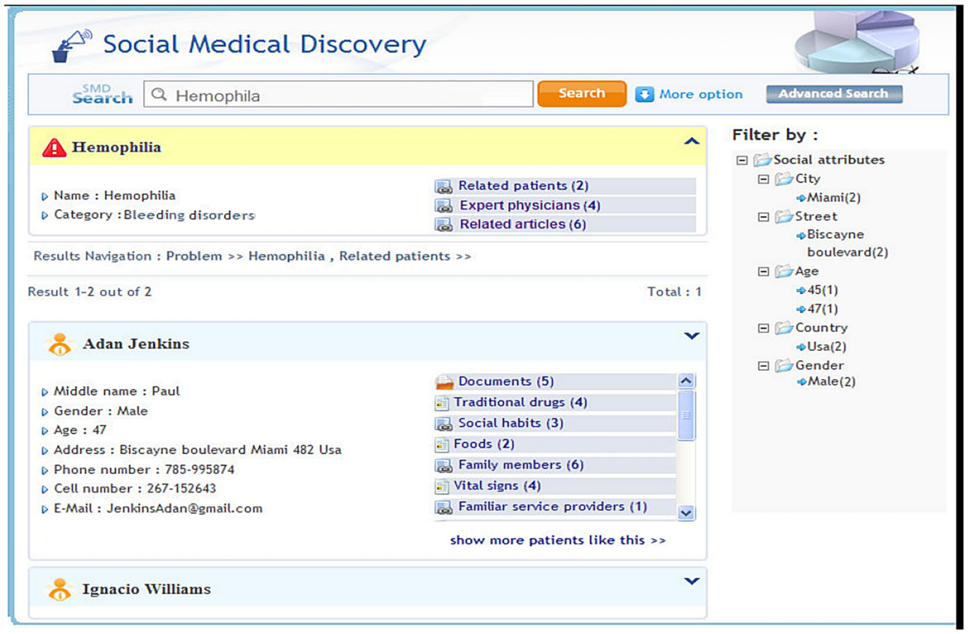
Interactive free-text queries allow users to simplify the way they express their information needs by using a seed query that can be refined by a set of subsequent query formulations, driven by user's goals and decisions. The main objective of such systems is to minimize the number of queries needed for answering the user needs.

Two popular approaches for interactive free-text queries exist, namely *query suggestion* [5, 8] and *faceted search* [30]. Search systems with query suggestion capabilities offer refinements to the user query [8], or provide keyword auto-completion services which help users to better express their information need, usually by utilizing queries previously submitted by other users [5]. Faceted search systems associate multiple classifications with queried entities (e.g., categories attached to text documents) which can be used later on for query refinement using a convenient faceted navigational scheme. Given the initial user query, faceted search systems retrieve both the set of relevant documents and a set of facets associated with the number of relevant documents per each facet value. Then, the user can refine the query by filtering the original result set based on one or more facets ("drilling-down").

Nowadays interactive free-text search is widely adopted by many discovery systems in various domains such as web search (e.g., Google Suggest), E-Commerce (e.g., eBay, Amazon, Shopping.com), (e.g., [2]), medical discovery systems [26], and many more.

Interactive structured queries follow the success of interactive free-text query systems. Several such systems have been proposed and provide either query suggestion [15] or faceted search interfaces [6]. Yet, similarly to one-shot structured query approaches, such systems are less popular than their interactive free-text query counterparts, due to the same reasons given for the one-shot case.

Recently, few works have identified the gap between the intuitiveness of interactive free-text queries and the usability


Social Medical Discovery

SMD Search | Hemophila | Search | More option | Advanced Search

Hemophilia

- Name : Hemophilia
- Category : Bleeding disorders

Related patients (2)
Expert physicians (4)
Related articles (6)

Results Navigation : Problem >> Hemophilia , Related patients >>

Result 1-2 out of 2 Total : 1

Adan Jenkins

- Middle name : Paul
- Gender : Male
- Age : 47
- Address : Biscayne boulevard Miami 482 Usa
- Phone number : 785-995874
- Cell number : 267-152643
- E-Mail : JenkinsAdan@gmail.com

Documents (5)
Traditional drugs (4)
Social habits (3)
Foods (2)
Family members (6)
Vital signs (4)
Familiar service providers (1)

show more patients like this >>

Ignacio Williams

Filter by :

- Social attributes
  - City
    - Miami(2)
  - Street
    - Biscayne boulevard(2)
  - Age
    - 45(1)
    - 47(1)
  - Country
    - Usa(2)
  - Gender
    - Male(2)

Figure 1: Social Medical Discovery – a search application in the healthcare domain that is based on our discovery system. Users can initiate an exploratory session by submitting a (free-text or structured) query and then follow a specific attribute of retrieved entities (shown in the *Filter by:* column) or one of their related entities (given within the patient's record). For privacy reasons all patient details are fake.

of interactive structured queries and proposed the concept of *incremental querying* [32] as a compromise. In an incremental querying setting, the user may start her search using simple keyword search. The search system then suggests the user a set of candidate structured queries, aiming at correctly translating the user needs. Then, the users can select appropriate structured query to be executed according to their needs.

ER discovery over unstructured data has recently attracted the attention of many researchers from a classical IR perspective. This extension follows the observation that for many user queries, entities are more suitable for query satisfaction than full documents such as web-pages or scientific papers. The INEX entity ranking track [14] studied entity ranking over Wikipedia articles where entities are assumed to correspond to Wikipedia entries. Example queries include "Italian Nobel prize winners", "Formula 1 drivers that won the Monaco Grand Prix", or "German spoken Swiss cantons". The TREC entity track [4] studied the *related entity finding* task over the ClueWeb collection, where the query specifies a source entity (usually the entity home-page), the type of target entities that should be retrieved, and the relationship type to consider. An example query for this task asks for teammates of "Michael Schumacher" when he was racing in Formula 1, with possible relevant answers being "Eddie Irvine" and "Felipe Massa" [3]. Popular approaches for handling such queries retrieve and rank entities that are mentioned or linked by relevant documents, as identified by traditional retrieval methods [4]. Several systems have been developed recently that extract entities and their inter-relationships from an unstructured or semi-structured collection of textual documents, providing an effective ER representation scheme using RDF and efficient SPARQL based entity search services [21, 31].

Proximity search: Entities and their inter-relationships, whether identified within a text corpora by named entities extraction tools, or provided explicitly, can be represented by Entity Relationship Graph (ERG). A graph node represents a unique entity in the system and an edge between two nodes specifies an existing relationship between the two. Measuring the proximity between entities, i.e., their relationship strength, or ranking entities according to their proximity to a given entity, are both fundamental retrieval tasks that were studied extensively, e.g., [20, 9, 22, 11]. Most solutions apply variants of a random walk over the graph. One weakness of the ERG model is the lack of a plain representation for non-binary entity relationships in the ER data model. Additionally, random walk computation is based on the entire graph hence it is very sensitive to any graph update.

Our Approach: In the context of related work, our exploratory system belongs to the class of *interactive structured queries*, as it supports facet search together with graph navigation. In addition, it provides a flexible query language that follows the *incremental querying* paradigm, spanning from free-text to structured queries. During the exploratory session, our system discloses only entities and relationships that are likely to be relevant to the user, according to previous queries, thus enabling effective data exploration.

3. MODEL

In this section we present the model building blocks of our entity-relationship search system. We first describe our entity-relationship model. We then describe our query model and show several example queries. We conclude this section with a description of our interactive query model.

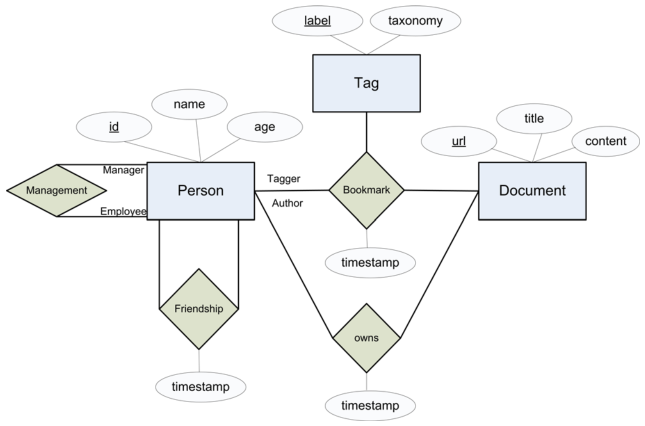


Figure 2: Example ER Model: Social Search

3.1 Entity-Relationship model

Our model is based on a simplified version of the conceptual entity-relationship model [10]. In this model, an entity e is defined as any object or "thing" that can be uniquely identified and distinguished from other entities. Each entity has a type $e.type$ used for its classification[2]. Each entity has a set of one to many attributes $e.a$ used to describe the entity's properties. Each attribute a has a name $a.name$, a type $a.type$ (e.g., String, Integer, Date) and a value $a.val$. An entity's key $e.key$ is further defined over its attributes, consisting of some minimal attribute subset that can be used to uniquely identify the entity. Each entity must have at least one key.

A relationship r captures an association between two or more entities, termed *relationship members*. Each relationship has a type $r.type$ and is uniquely identified by the combination of its entity member keys, denoted $r.key$. Each relationship entity member $r.e$ may have a role $r.e.role$ that captures the role this entity "plays" in the association. A relationship may further have zero to many attributes $r.a$, which can be used to describe its *context*.

3.1.1 Example: Social-search ER model

Figure 2 illustrates an example entity-relationship model related to the domain of social-search that we shall use to illustrate our main concepts. In this model, there are three main entity types, `Person`, `Document`, and `Tag`. Each entity has attributes that describe it and a key. For example, a person has an id (String) that further serves as its key, a name (String), and an age (Integer) attributes. Persons can own documents while their authorship role is further kept. Persons may bookmark documents with tags, and bookmarks are captured in the model by a ternary relationship `Bookmark` that connects the three entity types. For a person we further keep his tagger role. Each person may have friends captured by the `Friendship` relationship. Finally, the `Management` relationship captures management associations among persons, where for each person we keep either her manager or employee role.

3.2 Query model

A query over ER data can be used to discover entities based on patterns of interest, either by directly querying entities, their relationships, or both. A query result is a list of entities, possibly of various types, ranked according to their "relevance"[3] to the query.

Formally, a query q is a collection of one to many predicates. Each predicate describes some entity pattern, and the result of each predicate is expected to be a set of entities that "match" the specified pattern. Different predicates can be combined using the `AND`, `OR`, and `NOT` logical operators to construct a more complex query predicate. Complex predicates may be bounded with parentheses to denote their scope. Three types of query predicates are supported, namely, entity attribute predicates, free-text predicates, and entity relationship predicates. We now shortly explain each predicate type, accompanied with few examples.

3.2.1 Entity attribute predicates

An entity attribute predicate allows to query entities based on their type and attributes. The basic form of such predicate is:

```
entityType.attributeName:attributeValue
```

`entityType` defines a specific entity type or `*` for all possible types; `attributeName` defines a specific attribute name or `*` for all possible names; `attributeValue` defines a pattern of the attribute values of interest or `*` for all possible values. `attributeValue` can define either a specific value, or a range of possible values denoted as `[lowerBound TO upperBound]`, specifying a lower bound and upper bound (each bound can also be `*`).

The followings are several examples for entity attribute queries.

Example Query 1. *The query predicate `Person.*:*` (alternatively `Person.*` or `Person.`) can be used to return all person entities.*

A specific person entity may be further returned by using that entity's key; e.g., the query `Person.key:12345` will return a single person entity identified by the specified person key.

Example Query 2. *Return all person entities whose name is "John" and age is between 20 to 40:*

```
Person.name:"John" AND Person.age:[20 TO 40]
```

3.2.2 Free-text predicates

As already mentioned, issuing structured queries such as the aforementioned entity attribute queries requires some level of knowledge regarding structure. However, such knowledge is usually attained only by system experts, while the majority of users cannot translate the information need into structured queries. Free-text query predicates treat entities as text documents and therefore query the entities' content regardless of their structure. A free-text query predicate may include keywords, phrases, prefixes (e.g., ne*), etc. As an example, the query predicate `"CNN News"` (or `*.*:"CNN News"`) can be used to return entities that contain this phrase in at least one of their attribute values. The following example demonstrates how results can be restricted to a specific entity type.

[2]Multiple types can further encode inheritance hierarchies.

[3]We mostly focus in this work on retrieving entities that match the query. For some types of queries entities are ranked based on textual similarity to the query.

Example Query 3. *Return document entities related to "CNN News":*

```
Document.  AND "CNN News"
```

3.2.3 Relationship predicates

A relationship query predicate can be used to retrieve entities based on relationships they participate in. For that, a relationship predicate allows to define the entity relationship participation pattern, including the type of relationship, its relationship member patterns and their roles, and relationship attributes that limit the search according to some context. The result of a relationship query predicate is a set of entities that participate in the relationship subject to the relationship specified pattern.

More formally, a relationship predicate is defined by the following expression:

```
relType ( (WITH *) | (withExp (AND withExp)*))
```

`relType` specifies a specific relationship type or `*` if any type should be considered. `withExp` further denotes an expression that takes the form:

```
WITH (relMemPred | relAttPred)
```

`WITH` is a special operator used to describe a single relationship entity member pattern (`relMemPred`) or relationship attribute (`relAttPred`) pattern. `relMemPred` may include any combination of free-text or entity attribute predicates.

The following two queries demonstrate the usage of the `WITH` expression within a relationship predicate, once based on some relationship attribute, and once based on some relationship member pattern.

Example Query 4. *Return entities that participate in some Bookmark relationship with timestamp starting from 1/1/2011:*

```
Bookmark WITH timestamp:[1/1/2011 TO *]
```

Example Query 5. *Return entities that participate in some Bookmark relationship, where one of the relationship entity members is a tag that contains the keyword "news" as its label.*

```
Bookmark WITH Tag.label:news
```

It is worth noting that the result of the above query includes entities that are either tags that contain the keyword "news", persons that used such tags, and documents that were bookmarked with such tags. In order to further constrain the query result to a specific entity type, the above relationship predicate should be used in conjunction with a predicate that states the entity type of interest. For example, the following query extension obtains only documents:

Example Query 6.

```
Document.  AND (Bookmark WITH Tag.label:news)
```

Several `WITH` expressions can be combined together in the query to bind several relationship members and attributes, to form a complex relationship participation pattern.

Example Query 7. *Return entities associated with bookmarks that contain the tag "news" with documents that contain the expression "breaking news":*

```
Bookmark WITH Tag.label:news
         AND WITH Document.content:"breaking news"
```

Relationship members may be further constrained to a specific role in the relationship by using the `AS` clause.

Example Query 8. *Return person entities whose age is 25 or bellow and act as managers:*

```
Person.age:[* TO 25] AND
(Management WITH *) AND Person.  AS manager
```

3.3 Interactive Querying

The rich query model described above is mostly useful for expert users, not for novices. For a user who is not familiar with the query language or the data model, an interactive approach is preferred, where the information need of the user may be gradually covered by a series of query reformulations, starting from the user's original query, based on user selections.

More formally, an interactive query session is a final sequence of query reformulations $q_0, q_1, ..., q_k$, where q_0 is the initial query submitted by the user, q_k is the final query, and $q_i = \varphi(q_{i-1}, R_{i-1}, s_{i-1})$ for each $1 \leq i \leq k$. φ is a reformulation function, R_{i-1} is the set of entity results obtained by issuing q_{i-1}, and s_{i-1} represents the user's selection following the results from step $i-1$.

Our system supports two types of reformulation functions. The first type is *query based reformulation*, where the new query contains the previous query as a predicate. These reformulations are enabled using an extended faceted-search approach. In the ER model context, possible facets of interest are entity types, relationship types, and attribute names and values. The user selection of a category in the faceted search results adds a constraint to the previous query.

As an illustrative example, lets consider again Query 6. The information need expressed by this query could be equivalently achieved using the following interactive query session having 4 steps:

$q_0 =$ `"news"`
$s_0 =$ `entityType : Tag //focus on tags only`
$q_1 = \varphi(q_0, R_0, s_0) =$ `Tag.*:news`
$s_1 =$ `attributeName : label //focus on the tag's label attr.`
$q_2 = \varphi(q_1, R_1, s_1) =$ `Tag.label:news`
$s_2 =$ `relType : Bookmark //focus on Bookmark rel.`
$q_3 = \varphi(q_2, R_2, s_2) =$ `Bookmark WITH Tag.label:news`
$s_3 =$ `entityType : Document //focus on documents only`
$q_4 = \varphi(q_3, R_3, s_3) =$ `Document.  AND (Bookmark WITH Tag.label:news)`

While the previous example results in a conjunctive query, disjunctive queries can be also generated by further allowing users to have multiple facet selections.

The second reformulation type is *entity based reformulation*, where the user selects a single entity in the result set as a new query. Consider, for example, the following reformulation trail for discovering the manager of "John Doe":

$q_0 =$ `"John Doe"`
R_0 contains Person entity e_m with name "John Doe"
$s_0 = e_m,$ `relType : Management`
$q_1 = \varphi(q_0, R_0, s_0) =$ `Management WITH Person.key:12345`
$s_1 =$ `relationshipRole : manager`
$q_2 = \varphi(q_1, R_1, s_1) =$ `(Management WITH Person.key:12345) AND Person.  AS manager`

3.4 Discussion

While a wide range of information needs can be answered using the proposed interactive query language, there are still many queries which cannot be answered. In terms of query expressiveness on ER graphs, our query language supports arbitrary SPARQL-like star-queries; yet, multi-chain-queries cannot be easily answered [33]. Queries such as: "*return documents that were bookmarked by two friends*", which impose constraints on the relationship between result entities, are not supported directly by our query language and require additional efforts. We leave these extensions for future work.

4. IMPLEMENTATION

In this section we describe the implementation details of our ER search system. We first define a logical document model, and show how entity-relationship data is translated into this model. We then describe how the different query types presented in Section 3.2 can be evaluated using this implementation.

4.1 Logical document model

We hereby describe a logical document model used in the next section for representing entity-relationship data. Our logical document model consists of four main building blocks, namely, *Documents*, *Fields*, *Categories*, and *Category sets*. Each entity in the model is represented by a document while its attributes are represented by the document's fields. In addition, each entity is represented as a category, and a relationship is represented by a category set that contains all categories of participating entities. This dual representation allows efficient search for entities, either through their attributes as well as through their relationships with other entities, as we will show in the following.

Formally, a document d is defined by a quartet (id, F_d, C_d, CS_d); id is a unique document identifier; F_d is a collection of fields – each field $f \in F_d$ is defined by a triplet $(name, value, payload)$. C_d is a set of categories used to categorize the document. Each category $c \in C_d$ is defined by a path consisting a sequence of nodes $c = v_1/v_2/\ldots/v_l$ in the system taxonomy T. Finally, CS_d is a collection of category-sets, where each category-set $cs \in CS_d$ is defined by a set of pairs $cs = \{(c, payload) \mid c \in T\}$. The payload of a field or a category in a category-set, which is used to represent additional related information, is optional and not always required. In such cases we exclude it from the object's description.

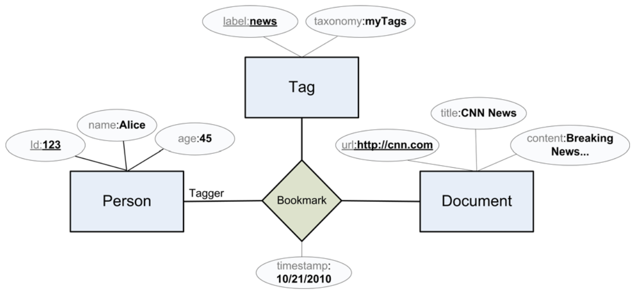
4.2 ER data representation

We now describe how entity-relationship data is represented in the logical document model. To illustrate this representation, an example of a `Bookmark` relationship instance with its entity members is given in Figure 3, together with a detailed example of the document constructed for the `Person` entity.

4.2.1 Entities

An entity e is represented by a document d_e, with F_e containing the following fields: 1) $f_{key} = (key, e.key)$ for the entity's key; 2) $f_{type} = (type, e.type)$ for the entity's type; 3) $f_a = (e.type|a.name, a.value, a.type)$, for each of the entity's attributes $e.a$; 4) $f_{a.name} = (attribute, a.name)$, for each of the entity's attributes. The last fields are required for supporting an efficient retrieval of all entities having these attribute names (e.g., **.age:**).

Additionally, to efficiently support free-text queries, a special field $f_{content} = (content, text)$, is added to d_e. The text



(a)

doc-id	1
F_e	$f_{key} = (key, \texttt{123})$ $f_{type} = (type, \texttt{Person})$ $f_{a_1} = (\texttt{Person\|id}, \texttt{"123",String})$ $f_{a_2} = (\texttt{Person\|name}, \texttt{"Alice",String})$ $f_{a_3} = (\texttt{Person\|age}, \texttt{45,Integer})$ $f_{a_1.name} = (attribute, \texttt{id})$ $f_{a_2.name} = (attribute, \texttt{name})$ $f_{a_3.name} = (attribute, \texttt{age})$ $f_{content} = (content, \texttt{"Alice. 45"})$ $f_{c_e} = (entity, key, \texttt{entity/Person/123})$
C_e	c_1=`entity/Person/123` c_2=`Integer/age/45` c_3=`relationship/Bookmark` c_4=`Date/timestamp/2010.10.21`
CS_e	$cs_1 = \{(c_1, tagger)$, `entity/Document/cnn.com`, `entity/Tag/news`, $c_3, c_4\}$

(b)

Figure 3: a) A *Bookmark* relationship with corresponding entities, and b) a logical document representing the *Person* entity.

value of this field is the concatenation of all attribute values represented as string literals. Some attributes may contain data which is irrelevant for free text search (e.g. the entity id). Therefore, the model allows defining for each attribute whether its string representation should be available for free-text search, and the text value of $f_{content}$ concatenates only those attributes' values. In Figure 3, the searchable content contains the person's name and age, and not the person's id.

As mentioned above, each entity e is represented as a category in the system taxonomy, with $c_e = entity/e.type/e.key$ as its path. In order to provide a mapping from d_e (the document representation of e) to c_e (its category representation), a special field $f_{c_e} = (entity, key, c_e)$ is added to F_e.

C_e includes categories which allow later on to perform faceted-search based on the entity type and attributes. Similarly to searchable attributes, one can decide which attributes worth also exposing as facets, representing attribute a as $c_a = a.type/a.name/a.value$. In Figure 3, the entity type and the person's age are further kept as categories.

4.2.2 Relationships

A relationship r is represented as a category-set cs_r that includes the categories of all relationship members. More specifically, for each relationship entity member $r.e$ with role $r.e.role$, the pair $(c_{r.e}, r.e.role)$ is added to cs_r. The category $c_{r.type} = relationship/r.type$ is also added to cs_r, to represent the relationship type *r.type*. As relationships can contain attributes, each relationship attribute *r.a* is represented by a category $c_{r.a} = r.a.type/r.a.name/r.a.value$, which is added to cs_r. In order to record relationship r for each of its entity members *r.e*, we add cs_r to CS_e.

As already mentioned, C_e includes all categories required for supporting faceted navigation, either through the relationship type, related entities, or relationship attributes. Therefore, in Figure 3, the categories of the relationship type and the bookmark timestamp attribute are further added to C_e.

4.3 Query Processing

In this section we describe the runtime process of each of the query types presented in Section 3.2. Recall that since an entity e has two representations (d_e and c_e), the set of entities which comply with a query can be generated either as a list of documents or as a list of categories. The transition from one representation to the other can be efficiently performed using f_{c_e} and f_{key}.

Most of the query processing described next is done using Lucene's *Term Query*, which accepts a term containing field name and a query term, and efficiently retrieves all documents in which this term appears in the context of the given field.

4.3.1 Free-text queries

Free-text queries are issued by applying Lucene's free-text search over the field $f_{content}$ which encapsulates all searchable content of the entities.

4.3.2 Entity and Attribute queries

When searching for entities of a specific type, only entity type is queried. The query is evaluated using the term query $(type, e.type)$. Thus, all entities of that type will be retrieved. When both entity type and attribute name and value are queried, i.e. looking for entities of a certain type *type* with a specific attribute a, we issue the term query $(type|a.name,\ a.value)$. If there is no constraint on the entity type, then, for each entity type in the system that includes the attribute *a.name*, a term query is created with this type and the given attribute as described above, and these term queries are combined into a Boolean OR query.

A special treatment is needed for queries which contain only attribute name. Such queries are evaluated using the term query $(attribute, a.name)$.

4.3.3 Relationship queries

Recall that a relationship query predicate allows to query entities based on their relationship participation pattern, including the relationship type, entity members whom they participate with, or relationship attributes that constrain the relationship context (see Section 3.2.3). This is done by binding the relationship type *r.type* with one or more `WITH` predicates. For example, looking back at Query 7, the constraints are on the relationship type *r.type* = `Bookmark`, a `WITH` predicate that constrains the relationship to include a `Tag` entity labeled with "news", and a `WITH` predicate that further constrains the relationship to include a `Document` entity having the "breaking news" expression within its content.

Note that while a candidate entity may satisfy all the relationship predicates independently, this does not imply that it also satisfies the relationship query. For example, a person that tagged a document containing "breaking news" with "report", and also tagged another document with the tag "news", satisfies all predicates of query 7 , however, it does not satisfy the query. Therefore, we need to examine each candidate based on additional validation of its corresponding category sets.

We start with evaluating each `WITH` predicate separately, and transforming each matching entity or attribute to its corresponding category. We collect all categories that match the predicate into a collection termed *predicate matching categories*. Next, for each predicate, we collect all category sets that contain at least one category from the predicate's matching categories. We term this collection *predicate matching category sets*. A category set then satisfies the relationship query if and only if it contains $c_{r.type}$ (i.e. it represents the desired relationship type), and is included in all predicate matching category sets (i.e., it satisfies all `WITH` predicates).

The final result of the process is a list of category sets that satisfy the relationship query. The entities which take part in each of these category sets are obtained using the category to entity mapping, and reported as query results.

5. EXAMPLE APPLICATION

We now shortly describe an example application that was implemented using the solution proposed in this paper named "Social-Medical Discovery" (SMD); an extension of social search for the medical domain which is part of the IBM Patient Empowerment System (IBM PES). IBM PES (whose social-medical discovery user interface is depicted in Figure 1) is a novel clinical decision support system. IBM PES *empowers the patients* and helps increasing patient safety by assisting patients and their medical providers with daily medical decision-making.

Many of IBM PES services require to uniformly search social and medical data, stored in its heterogeneous data repositories, in order to gain insights and provide value. We term such services as *social-medical discovery* services [25]. Example services span from simple search services that require to locate relevant information about some patient or medication, to more complex data exploration services that require to query the social-medical "dataspace" to reveal interesting patterns (e.g., relevant patients for some new clinical trial). Using our solution, even non-expert users of IBM PES, such as patients and physicians naturally are, can submit queries over multitude of social-medical data sources, regardless of their type, underlining data model, and semantics.

A typical search in IBM PES starts with the initial user need, usually expressed as a free text query. Then, users can utilize IBM PES interactive query interface for guiding their search towards achieving their information seeking goals. Every interactive querying step results in a query reformulation that adds structured query predicates that best capture the user's information need.

As an example, lets assume a searcher that submits an initial text query "`Warfarin 20mg`". As a result to her query, the system returns all social and medical entities that include these keywords in the "content" given by their attributes. First, the searcher can choose the facet `entity-Type:Medication` to refine her query to return only related medication entities. Next, the searcher may be interested in exploring patients that consume some specific medication by clicking on the `relType:PatientMed` relationship facet. Finally, the searcher may be interested in limiting her search to return only patients whose age is between 20 to 30 by defining an age range using the `attributeName:age` facet. The final query that represents the searcher information seeking goal would result with the following query written in our query language:

```
Patient.age:[20 TO 30] AND
(PatientMed WITH Medication.*:"Warfarin 20mg"))
```

6. EVALUATION

We now present the experimental results of an evaluation study conducted with our system. We first describe the datasets used in this study based on a benchmark recently published by Coffman et al. [12] (termed Coffman's benchmark hereinafter). We then present our experimental setup and the evaluation scheme. We first compare the quality of our system with that of two other discovery approaches. We then provide a more fine-granular, query-type level, quality analysis. We conclude this section with runtime analysis, demonstrating that our system is both efficient and effective.

6.1 Datasets

The Coffman's benchmark was used in [12] for evaluating the quality of several state-of-the-art database keyword search systems. The benchmark includes three different datasets, namely Mondial, IMDB, and Wikipedia. Each dataset is given in the form of database tables accompanied with a database schema that describes its structure. We converted each dataset into rich entity-relationship data by utilizing the primary-key and foreign-key constraints given in each dataset schema for "reverse-engineering" of the underlying conceptual ER model. Table 1 provides the details of each dataset, including size, number of tuples, and the number of entities and relationships after the conversion to ER model.

The Mondial dataset [24] includes geographical and demographic information from the CIA World Factbook, the International Atlas, the TERRA database, and other web sources. The Mondial dataset is fully structured and does not include rich text values.

The IMDB dataset is based on the Internet Movie Database website (*http://www.imdb.com/*) and includes information about movies, actor, directors, etc. While the underlying data model of the IMDB dataset is structured, this dataset further includes for each movie a special rich text attribute (Title.info) that provides some related content, e.g., parts from the movie's screenplay. Therefore, IMDB serves as an "hybrid" dataset having a structured data model accompanied with additional rich content.

The Wikipedia dataset includes data extracted from about 5500 articles chosen for the 2008-2009 Wikipedia Schools DVD, associated with revising users and other linked articles. In comparison with the two other datasets, this dataset is mostly textual and has a relatively simple data model.

6.2 Experimental setup and evaluation

For evaluation, we utilized the set of topics that are provided in Coffman's benchmark. For each dataset, there are 50 topics, each is expressed by free-text and is associated

Dataset	Size (MB)	Tuples	Entities	Relationships
Mondial	9	17,115	6,004	11,198
IMDB	516	1,673,074	454,740	812,695
Wikipedia	550	206,318	7,781	193,491

Table 1: Datasets details

with a qrels (query relevance set) that includes a list of correct tuple answers, from which we obtained the list of relevant entities. The complexity of the given topics ranges from the demand to query specific entity attributes, to more "sophisticated" topics that require the combination of several relationship queries to obtain the correct answers [12].

Using our search system's query language, we managed to generate queries for the full set of 150 topics. It is important to note that each such query corresponds to the final query in a faceted search navigation session that "best" covers the information need as given in Coffman's benchmark.

We compared our system with two other approaches that were presented in Section 1, namely (one-shot) free-text search and faceted-search. Search quality of the various solutions was measured using mean-average-precision (MAP) computed at cutoff 1000.

For free-text search, we took the original text-queries, from each dataset, and submitted them as is to the Lucene index. For that, each entity was represented in the index by a single document with its content obtained by "scraping" all the textual values and metadata associated with it.

As for faceted-search, we used a state-of-the-art faceted-search system [7]. In such a system we can only index documents and their associated categories [7]. Therefore, while we could easily index entities as documents and use their categories to capture entity types and attributes, there is no straight-forward way to fully capture their relationships with other entities. Similarly to the query generation for our approach, we generated the facet queries such that each query corresponds to the final query in a faceted search navigation session that "best" covers the information need.

6.3 Results

We now report the results of the quality analysis of our search system (denoted `ER-FS`) and its comparison with the two other approaches using Coffman's benchmark, namely, one-shot free-text search (denoted `free-text`) and faceted-search (denoted `FS`). We further report on the query runtime analysis for each dataset.

6.3.1 Quality comparison

Figure 4 depicts the result of our comparison between the various approaches per queried dataset. First, we clearly observe that for all datasets our system obtains higher MAP than that of other approaches. The relative quality improvement of our system over the next best performing approach was about 66%, 53.94%, and 5.78%, for the Mondial, IMDB, and Wikipedia datasets respectively, with a significant improvement for the first two datasets: $p < 10^{-7}$ for Mondial dataset and $p < 10^{-5}$ for IMDB dataset (sign test). For the Wikipedia dataset the improvement was not significant, probably because only five topics in this dataset require the advanced capabilities of relationship querying which are not supported by the second performing `FS` approach.

A closer look at the performance trends of our `ER-FS` approach compared to those of the simple `free-text` approach

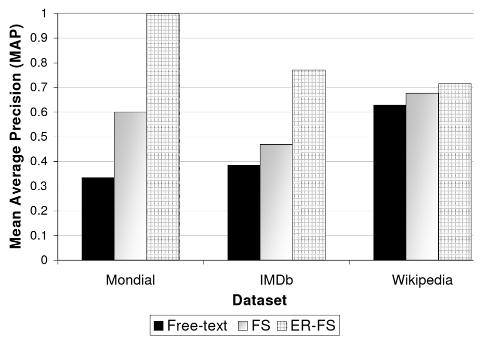


Figure 4: MAP values obtained by our system (ER-FS) for each dataset compared to the MAP values obtained by the free-text and faceted search (FS).

in Figure 4 reveals an interesting observation. Moving from the fully structured Mondial dataset to the semi-structured Wikipedia dataset, we observe that using the same querying mechanism `ER-FS` can answer structured queries over fully structured data with high quality guarantees compared to the other approaches, as well as answer a wide range of free-text queries over semi-structured data. This serves as a strong testament for the capability of our search system to provide both flexible and intuitive querying over a broad range of entity-relationship data.

6.3.2 *Quality analysis*

We now report on a more fine-granular quality analysis of our search system, further breaking the analysis according to the type of queries in each dataset in Coffman's benchmark. We identify five different types of queries that were generated over all datasets, namely, *free-text queries* that query entities based on their searchable attributes ignoring the entity's metadata or data structure, *single entity attribute predicates* that query entities directly based on their properties, *single relationship predicates* that query entities based on their participation in some given relationship, *multiple relationship predicates* that combine several relationship query predicates, and *hybrid queries* that combine both entity attribute query predicates and relationship query predicates.

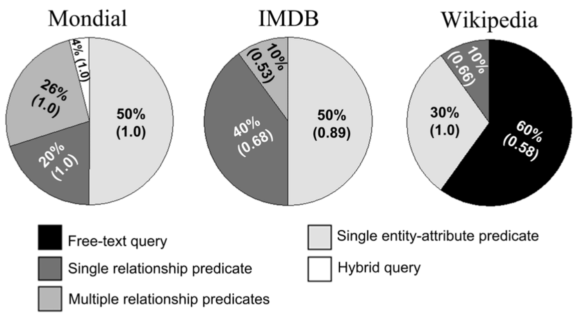


Figure 5: Query type distribution for each queried dataset using our query language. For each dataset and query type, MAP values obtained by our system are further reported in brackets.

Figure 5 depicts for each dataset a pie-chart that describes its query type distribution. For each query type and dataset, we further report on the relative quality obtained by the set of queries that belong to that type (marked in brackets). First, as expected, we can observe that as we move from Wikipedia's semi-structured dataset to the Mondial's fully structured dataset, our system uses more structured query predicates and less text-based queries. We can also observe that the relative complexity of the queries increases as we move to a more structured queried dataset.

The fine-granular quality analysis per query type shades some light on the reasons for the better performance of our system compared to the other approaches. For the Mondial dataset we observe that our search system completely manages to correctly answer all types of queries (hence achieving MAP=1.0). This is not surprising given that the Mondial dataset is fully structured and our query language can fully express the information needs given in this dataset.

For the IMDB dataset we observe that as the query complexity increases the performance decreases, ranging from 0.89 MAP value for single entity attribute predicate queries, to 0.53 MAP value for multiple relationship predicate queries. Although such decrease in quality for complex queries is expected, our system manages to provide almost 54% improvement in performance over the second best approach.

Finally, and most interesting, 60% of the queries in the Wikipedia dataset can be answered using simple free-text queries. Therefore, improved quality for the other 40% (20 queries) is directly attributed to the ability to refine the original free-text queries according to the data structure of Wikipedia's ER model. As shown in Figure 4, both `FS` and `ER-FS` provide an improvement on top of the quality obtained by the `free-text` approache. Furthermore, `ER-FS` provides better quality for five queries in the benchmark compared to the `FS` approach, due to its unique ability to serve relationship queries, a capability that `FS` does not have.

6.3.3 *Runtime Analysis*

We further analyzed the efficiency of our search system in terms of query runtime by repeated execution of Coffman's benchmark queries several times while recording the average query runtime and standard deviation for each dataset.

Additionally, we further used the Yago ontology [29] in order to evaluate the query runtime of our system over a dataset of a relatively large scale. Yago is a large collection (6GB) of facts extracted from both Wikipedia and WordNet containing millions of facts about people, places, etc. We translated the Yago ontology from Notation3 format (*http://www.w3.org/DesignIssues/Notation3*) into our ER data model, resulting in about 1M entities and 500K relationships. The Yago dataset includes facts that correspond to the queries in Coffman's benchmark. Therefore, we expressed the queries in Coffman's benchmark using our Yago data model and submitted the complete set of 150 queries against our Yago generated search system, measuring repeated query runtime.

We have run the experiments on a Windows server machine with 4GB memory. The results of the runtime analysis are depicted in Table 2. As one can observe, query time increases with dataset size and complexity. However, the query runtime remains sub-second independently of the dataset size, its type (i.e., fully structured or semi-structured), or the query type. This demonstrates the efficiency of our system, in addition to its effectiveness.

Dataset	Size (MB)	Query runtime (msec)
Mondial	9	22.99(±4.00)
IMDB	516	482.19(±14.31)
Wikipedia	550	119.99(±4.29)
Yago	5908	334.94(±89.62)

Table 2: Query runtime analysis

7. SUMMARY

In this paper we described a novel discovery approach over rich entity-relationship data. We presented its fundamentals, based on a unique combination of expressive, yet intuitive, query language, faceted search, and graph navigation. We provided a detailed description of our solution which extends an existing facet search library for indexing and searching generalized entity-relationship data models, allowing new exploration capabilities. We then evaluated our system by comparing its performance to two other existing approaches and demonstrated its effectiveness and efficiency.

Our work can be extended in several ways. First, we wish to extend our interactive query language to support more query types, e.g., SPARQL-like multi-chain-queries [33]. Second, though the experimental results demonstrate that our ER discovery approach performs relatively well, there is still a lot of room for improvement, especially for semi-structured data. Our search system still depends on Lucene's own ranking mechanism for ranking entities. For future work we intend to explore novel entity ranking methods, including proximity and context-aware ranking models. Finally, we wish to study the usability of the suggested exploration approach using user studies to better understand the ability of users with different levels of expertise to interact with our ER search system. We believe that the new discovery approach described in this work opens many new challenges in terms of human computer interaction.

8. REFERENCES

[1] B. Aditya, Gaurav Bhalotia, Soumen Chakrabarti, Arvind Hulgeri, Charuta Nakhe, Parag Parag, and S. Sudarshan. Banks: browsing and keyword searching in relational databases. In *Proceedings of VLDB*, pages 1083–1086. VLDB Endowment, 2002.

[2] Einat Amitay, David Carmel, Nadav Har'El, Shila Ofek-Koifman, Aya Soffer, Sivan Yogev, and Nadav Golbandi. Social search and discovery using a unified approach. In *Proceedings of Hypertext and Hypermedia*, pages 199–208. ACM, 2009.

[3] Krisztian Balog, Edgar Meij, and Maarten de Rijke. Entity search: building bridges between two worlds. In *Proceedings of SEMSEARCH*, pages 9:1–9:5. ACM, 2010.

[4] Krisztian Balog, Pavel Serdyukov, Arjen P. De Vries, Paul Thomas, and Thijs Westerveld. Overview of the TREC 2009 entity track. In *Proceedings of TREC*, 2009.

[5] Holger Bast and Ingmar Weber. Type less, find more: fast autocompletion search with a succinct index. In *Proceedings of SIGIR*, pages 364–371. ACM, 2006.

[6] Senjuti Basu Roy, Haidong Wang, Gautam Das, Ullas Nambiar, and Mukesh Mohania. Minimum-effort driven dynamic faceted search in structured databases. In *Proceeding of CIKM*, pages 13–22. ACM, 2008.

[7] Ori Ben-Yitzhak, Nadav Golbandi, Nadav Har'El, Ronny Lempel, Andreas Neumann, Shila Ofek-Koifman, Dafna Sheinwald, Eugene Shekita, Benjamin Sznajder, and Sivan Yogev. Beyond basic faceted search. In *Proceedings of WSDM*, pages 33–44. ACM, 2008.

[8] Huanhuan Cao, Daxin Jiang, Jian Pei, Qi He, Zhen Liao, Enhong Chen, and Hang Li. Context-aware query suggestion by mining click-through and session data. In *Proceeding of SIGKDD*, pages 875–883. ACM, 2008.

[9] Kaushik Chakrabarti, Venkatesh Ganti, Jiawei Han, and Dong Xin. Ranking objects based on relationships. In *Proceedings of SIGMOD*, pages 371–382. ACM, 2006.

[10] Peter Pin-Shan Chen. The entity-relationship model-toward a unified view of data. *ACM Trans. Database Syst.*, 1:9–36, March 1976.

[11] Tao Cheng, Xifeng Yan, and Kevin Chen-Chuan Chang. Entityrank: searching entities directly and holistically. In *Proceedings of VLDB*, pages 387–398. VLDB Endowment, 2007.

[12] Joel Coffman and Alfred C. Weaver. A framework for evaluating database keyword search strategies. In *Proceedings of CIKM*, pages 729–738. ACM, 2010.

[13] Joel Coffman and Alfred C. Weaver. Structured data retrieval using cover density ranking. In *Proceedings of the Workshop on Keyword Search on Structured Data*, pages 1:1–1:6. ACM, 2010.

[14] Gianluca Demartini, Tereza Iofciu, and Arjen P. De Vries. Overview of the INEX 2009 entity ranking track. In *Proceedings of the INEX'09*, pages 254–264. Springer-Verlag, 2010.

[15] Carlos Garcia-Alvarado, Zhibo Chen, and Carlos Ordonez. Olap-based query recommendation. In *Proceedings of CIKM*, pages 1353–1356. ACM, 2010.

[16] Lin Guo, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram. Xrank: ranked keyword search over xml documents. In *Proceedings of SIGMOD*, pages 16–27. ACM, 2003.

[17] Alon Y. Halevy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *Proceedings of PODS*, pages 95–104, 1995.

[18] Andreas Hotho, Robert Jäschke, Christoph Schmitz, and Gerd Stumme. Information retrieval in folksonomies: Search and ranking. In *Proceedings of ESWC '06*, pages 411–426, 2006.

[19] Vagelis Hristidis and Yannis Papakonstantinou. Discover: keyword search in relational databases. In *Proceedings of VLDB*, pages 670–681. VLDB Endowment, 2002.

[20] Glen Jeh and Jennifer Widom. SimRank: a measure of structural-context similarity. In *Proceedings of SIGKDD*, pages 538–543. ACM Press, 2002.

[21] Gjergji Kasneci, Fabian M. Suchanek, Georgiana Ifrim, Maya Ramanath, and Gerhard Weikum. Naga: Searching and ranking knowledge. In *Proceedings of ICDE*, pages 953–962. IEEE Computer Society, 2008.

[22] Yehuda Koren, Stephen C. North, and Chris Volinsky. Measuring and extracting proximity in networks. In *Proceedings of SIGKDD*, pages 245–255, New York, NY, USA, 2006. ACM.

[23] Yuangui Lei, Victoria Uren, and Enrico Motta. SemSearch - a search engine for the semantic web. 2006.

[24] W. May. Information extraction and integration with florid: The mondial case study. *Technical Report 131*, Universität Freiburg, Institut für Informatik.

[25] Haggai Roitman, Yossi Messika, Yevgenia Tsimerman, and Sivan Yogev. A unified approach for social-medical discovery. In *Proceedings of the 23rd International Conference of the European Federation for Medical Informatics (MIE)*, Oslo, Norway, 2011.

[26] Haggai Roitman, Sivan Yogev, Yevgenia Tsimerman, Dae Won Kim, and Yossi Messika. Exploratory search over social-medical data. In *Proceedings of CIKM*, 2011.

[27] Ian Ruthven. Interactive information retrieval. *Annual Rev. Info. Sci & Technol.*, 42:43–91, January 2008.

[28] SPARQL. http://www.w3.org/tr/rdf-sparql-query/.

[29] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of WWW*, 2007.

[30] Daniel Tunkelang. *Faceted Search*. Morgan & Claypool Publishers, 2009.

[31] Gerhard Weikum, Gjergji Kasneci, Maya Ramanath, and Fabian Suchanek. Database and information-retrieval methods for knowledge discovery. *Commun. ACM*, 52:56–64, April 2009.

[32] Gideon Zenz, Xuan Zhou, Enrico Minack, Wolf Siberski, and Wolfgang Nejdl. From keywords to semantic queries-incremental query construction on the semantic web. *Web Semant.*, 7:166–176, September 2009.

[33] Qi Zhou, Chong Wang, Miao Xiong, Haofen Wang, and Yong Yu. Spark: adapting keyword query to semantic search. In *Proceedings of ISWC/ASWC*, pages 694–707. Springer-Verlag, 2007.