NAGA: Searching and Ranking Knowledge

Gjergji Kasneci Fabian M. Suchanek Georgiana Ifrim Maya Ramanath Gerhard Weikum
Max-Planck Institute for Informatics
Saarbrücken, Germany
{kasneci, suchanek, ifrim, ramanath, weikum}@mpi-sb.mpg.de

Abstract— **The Web has the potential to become the world's largest knowledge base. In order to unleash this potential, the wealth of information available on the Web needs to be extracted and organized. There is a need for new querying techniques that are simple and yet more expressive than those provided by standard keyword-based search engines. Searching for knowledge rather than Web pages needs to consider inherent semantic structures like entities (person, organization, etc.) and relationships (isA, locatedIn, etc.).**

In this paper, we propose NAGA, a new semantic search engine. NAGA builds on a knowledge base, which is organized as a graph with typed edges, and consists of millions of entities and relationships extracted from Web-based corpora. A graph-based query language enables the formulation of queries with additional semantic information. We introduce a novel scoring model, based on the principles of generative language models, which formalizes several notions such as confidence, informativeness and compactness and uses them to rank query results. We demonstrate NAGA's superior result quality over state-of-the-art search engines and question answering systems.

I. Introduction

The World Wide Web bears the potential of being the world's most comprehensive knowledge base, but we are far from exploiting this potential. The Web includes a wild mixture of valuable scientific and cultural content, news and entertainment, community opinions, advertisements, as well as spam and junk. Unfortunately, all this is coiled up into an amorphous pile of hyperlinked pages, and keyword-oriented search engines merely provide best-effort heuristics to find relevant "needles" in this humongous "haystack".

As a concrete example, suppose we want to learn about physicists who were born in the same year as Max Planck. First, it is close to impossible to formulate this query in terms of keywords. Second, the answer to this question is probably distributed across multiple pages, so that no state-of-the-art search engine will be able to find it, and third, the keywords "Max Planck" could stand for different world entities (e.g., the physicist Max Planck, the Max-Planck Society, etc.). In fact, posing this query to Google (by using the keywords "physicist born in the same year as Max Planck") yields only pages about Max Planck himself, along with pages about the Max-Planck Society. This example highlights the need for more explicit, unifying structures for the information of the Web. A knowledge base which could understand binary predicates, such as Max_Planck isA physicist or Max_Planck bornInYear 1858 would go a long way in addressing information needs such as the above. Combined with an appropriate query language and ranking strategies, users would be able to express queries with semantics and retrieve precise information in return.

There are several research avenues that aim at this direction in a broader sense. Large-scale information extraction from semistructured corpora or unstructured text sources has made great progress in recent years [1], but it is not addressing the *querying* of the acquired knowledge. Graph querying such as RDF-based languages or data mining on biological networks is a direction that is gaining momentum [17], but does not consider the potential uncertainty of the data and disregards the need for a ranking model. Finally, entity-oriented Web search and other forms of "semantic" information retrieval [8] provide ranking but have rather simple query models such as keyword search. Ranked retrieval on XML data like XQuery Full-Text are more expressive but focus on trees and do not carry over to richer knowledge graphs. Our work positions itself at the confluence of these research avenues and creates added value by combining techniques from all of them and further extending this synergetic approach by various novel building blocks.

Our approach: In this paper, we describe NAGA, a new semantic search engine. NAGA's data model is a directed, weighted, labeled multi-graph $G = (V, E, L_V, L_E)$. V is a set of nodes, $E \subseteq V \times V$ is a multi-set of edges, L_V is a set of node labels and L_E is a set of edge labels. Each node $v \in V$ is uniquely assigned a label $l(v) \in L_V$ and each edge $e \in E$ is assigned a label $l(e) \in L_E$. Each node represents an entity and each edge represents a relationship between two entities. For example, the edge $e = (v, w)$ with $l(e) =$ bornInYear, $l(v) =$ Max_Planck(physicist), and $l(w) =$ 1858 represents the fact that the physicist Max Planck was born in 1858. We simply write Max_Planck(physicist) bornInYear 1858.

For each fact f, we maintain all URLs of Web pages in which f occurred and refer to these pages as the *witnesses* of f. Furthermore, each fact f is assigned a *confidence value* that depends on the estimated *accuracy* $acc(f, p)$ with which the fact f was extracted from a witness p and the *trust* $tr(p)$ we have in p. While the accuracy value is usually provided by the extraction mechanism the trust in a witness can be computed by any algorithm similar to PageRank. Suppose that a fact f was extracted from the witnesses $p_1, \ldots, p_n$. We add f only once to the knowledge graph and compute its confidence value as:

$$c(f) = \frac{1}{n} \sum_{i=1}^{n} acc(f, p_i) \cdot tr(p_i) \qquad (1)$$

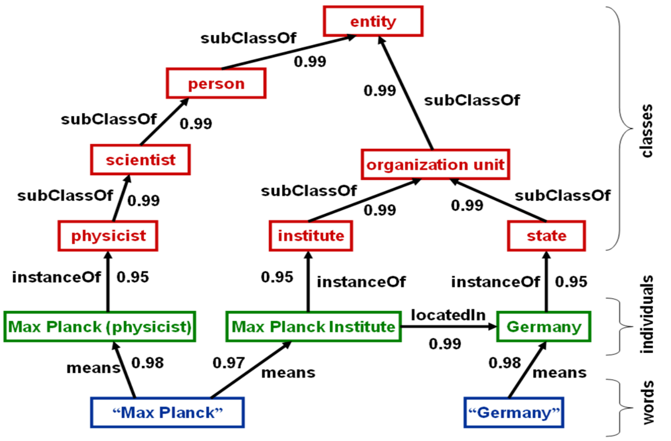

Fig. 1. Excerpt from the knowledge graph

NAGA's knowledge base currently consists of 16 million facts extracted from semi-structured Web-based sources such as Wikipedia and IMDB as well as hand-crafted ontologies such as WordNet [14]. Additionally, we utilize state-of-the-art extraction tools such as LEILA [29] in order to extract facts from unstructured Web-pages containing natural language text. As of now, NAGA understands 26 predefined relationships such as isA, bornInYear, establishedInYear, hasWonPrize, locatedIn, politicianOf, means, actedIn, discoveredInYear, discoveredBy, isCitizenOf, before, after, etc. Figure 1 depicts an excerpt from the knowledge graph. For more detailed information on the construction of the knowledge graph please refer to [30], [29].

In order to query the knowledge-graph, NAGA provides a graph-based query language. The query language allows the formulation of queries with semantic information. For example, Figure 2 shows how the preceding query about physicists born in the same year as Max Planck can be formulated with explicit semantic information (e.g. "Max Planck" is a physicist). NAGA also allows more complex graph queries with regular expressions over relationships as edge labels.

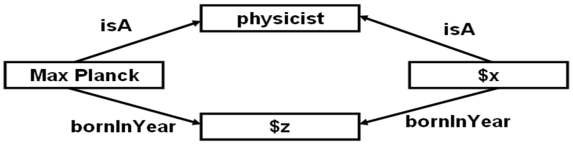

Fig. 2. Example query

NAGA returns multiple answers for some queries. In order to rank these answers, we propose a novel scoring mechanism based on the principles of generative language models for document-level information retrieval [25], [31]. We apply these principles to the specific and unexplored setting of weighted, labeled graphs. Our scoring model is extensible and tunable and takes into consideration several intuitive notions such as compactness, informativeness and confidence of the results.

Contributions: Our major contributions in this paper are:

1) a novel, expressive yet concise query language for searching a Web-derived knowledge base,
2) a novel ranking model based on a generative language model for queries on weighted and labeled graphs,
3) an extensive evaluation of the search-result quality provided by NAGA, based on user assessments and in comparison to state-of-the-art search engines and question answering systems like Google, Yahoo! Answers, and START [23].

Furthermore, we demonstrate the superiority of NAGA's ranking mechanism over comparable mechanisms as used in [6], [22].

The rest of the paper is organized as follows. Section II describes the query language with several examples of its use. In Section III, we present our novel scoring model. Section IV discusses the query processing algorithms, and Section V presents the experimental evaluation of our system. We discuss related work in Section VI before concluding in Section VII.

II. QUERY LANGUAGE

A. Formal Query Model

Given a set S of labels (e.g. relation labels), we denote by $\text{RegEx}(S)$ the set of regular expressions over S. We write $\mathcal{L}(r)$ ($\subseteq S^*$) to denote the language of some $r \in \text{RegEx}(S)$.

Definition 2.1 (Query): A query *is a connected directed graph $Q = (V, E, L_V, L_E, U)$, where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, L_V is a set of word, individual and class labels, L_E is a set of relation labels, and U is a set of variables. Each vertex $v \in V$ is assigned a label $l(v) \in L_V \cup U$. Each edge $e \in E$ is assigned a label $l(e) \in \text{RegEx}(L_E) \cup U \cup \{$connect$\}$. If an edge or vertex is labeled with a variable, we call that edge or vertex* unbound*. We disallow unbound edges between two unbound vertices.*

As in the knowledge graph, the vertex labels L_V denote entities and the edge labels L_E denote relations. Edges can be labeled by the special keyword connect or by a regular expression over L_E. We say that an edge is labeled by a *simple relation* if its label is contained in L_E. In addition, the labels of nodes and edges can be variables. We call an edge of a query graph a *fact template* and denote it by its edge label and the two node labels. For example, Albert_Einstein friendOf $x is a fact template. Here, $x denotes a variable.

NAGA's answer model is based on graph matching. Given a query, NAGA aims to find subgraphs of the knowledge graph that match the query graph. We say that a vertex v from a knowledge graph *matches* a query vertex with label $\mathbf{l}$, if $l(v) = \mathbf{l}$ or if $\mathbf{l}$ is a variable. Furthermore, we say that a query vertex v' is *bound* by a vertex v of the knowledge graph if v matches v'. Before defining matches to our queries, we first define matches to fact templates.

Definition 2.2 (Matching Path): A matching path *for a fact template x r y is a sequence of edges $m_1, \ldots, m_n$ from the knowledge graph, such that the following conditions hold:*

- *If r is a variable, then $n = 1$ and the start node of m_1 matches x and the end node of m_1 matches y.*
- *If r is a regular expression, then $m_1, \ldots, m_n$ forms a directed path and $l(m_1) \ldots l(m_n) \in \mathcal{L}(r)$. Furthermore, the start node of m_1 matches x and the end node of m_n matches y*[1].
- *If r=*connect*, then $m_1, \ldots, m_n$ forms an undirected path, such that its start node matches x and its end node matches y.*

Given a query q and an answer graph g, we denote the matching path of a query template q_i from q by $match(q_i, g)$. Now we generalize this definition to queries:

Definition 2.3 (Answer Graph): An answer graph *to a query q is a subgraph A of the knowledge graph, such that (1) for each fact template in q there is a matching path in A, (2) each fact in A is part of one matching path, (3) each vertex of q is bound to exactly one vertex of A.*

We will occasionally use the label isA as a shorthand for the regular expression instanceOf subclassOf*. isA connects an individual via one instanceOf-edge to its immediate class and by several subclassOf-edges to more general superclasses.

B. Query types

We provide a taxonomy of three query types in ascending order of expressiveness:

Discovery Queries are queries that supply the user with pieces of missing information. For example, Figure 2 asks for *physicists who were born in the same year as Max Planck*. NAGA attempts to fill in the variables by finding a subgraph in the knowledge graph that matches the query and thus binds the two variables. Note that there are multiple answers to this query and NAGA returns a ranked list of answers. One possible answer is shown in Figure 3.

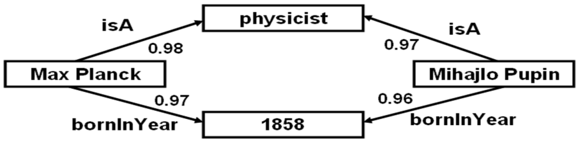

Fig. 3. Answer to query of Figure 2

Formally, a discovery query is a query in which at least one fact template has an unbound component.

Regular Expression Queries enable users to specify more flexible matches by allowing *regex labels* on query edges. Figure 4 shows two queries, the first of which uses the regular expression locatedIn* and the shorthand isA to ask *Which rivers are located in Africa?* Here, an answer such as Nile instanceOf river, Nile locatedIn Egypt, Egypt locatedIn Africa is a valid match. The second query asks for *Scientists whose first name or last name is Liu*.

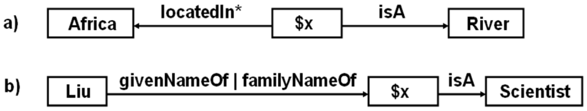

Fig. 4. Regular expression query examples

We say that a regular expression query is a query in which at least one edge is labeled by a regular expression that is not a simple relation.

Relatedness Queries discover "broad" connections between pieces of information. For example, Figure 5 asks the question *How are Margaret Thatcher and Indira Gandhi related?* There are several possible answers to this query – including the trivial answer that "they are both people", more informative answers such as "they were both prime-ministers" as well as more complex answers such as "Margaret Thatcher and Indira Gandhi were both prime-ministers of English-speaking countries (England and India)".

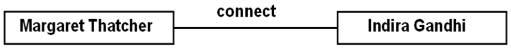

Fig. 5. Relatedness query example

Formally, a relatedness query is a query in which at least one edge is labeled by the connect label.

NAGA returns multiple answers for some queries. In order to rank these answers, we propose a novel scoring mechanism.

III. Ranking Answer Graphs

A good ranking model for answer graphs should satisfy the following desiderata:

1) *Confident* answers (i.e. answers containing facts with high extraction confidence from authoritative pages) should be ranked higher.
2) *Informative* answers should be ranked higher. For example, when asking a query Albert_Einstein isA $z the answer Albert_Einstein isA physicist should rank higher than the answer Albert_Einstein isA politician, because Einstein is rather known as a physicist than as a politician. Similarly, for a query such as $y isA physicist, the answers about world class physicists should rank higher than those about hobby physicists.
3) *Compact* answers should be preferred, i.e. direct connections rather than loose connections between entities are preferable. For example, for the query *How are Einstein and Bohr related?* the answer about both having won the Nobel Prize should rank higher than the answer that Tom Cruise connects Einstein and Bohr by being a vegetarian

[1] For the special case $r \in L_E \subseteq \text{REGEX}(L_E)$ (in which r is a simple relation) the match is an edge and $n = 1$.

like Einstein, and by being born in the year in which Bohr died.

We propose a novel scoring model that integrates all the above desiderata in a unified framework. Our approach is inspired by existing work on language models (LM) for information retrieval (IR) on document collections [31], [19], but it is adapted and extended to the new domain of knowledge graphs. In this setting, the basic units are not *words*, but *facts* or *fact templates*. Our graphs and queries can be seen as sets of facts or fact templates respectively. A candidate result graph in our setting corresponds to a document in the standard IR setting.

The language model we propose is much more challenging than the traditional language models for two reasons:

1) By considering facts and fact templates as IR units rather than words-in-documents, our queries include both bound and unbound arguments - a situation that is very different from what we encounter in multi-term queries on documents.
2) Our corpus, the knowledge graph, is virtually free of redundancy (each fact occurs only once), unlike a document-level corpus. This makes reasoning about background models and idf-style aspects [31] more subtle and difficult.

A. Language Model

In line with IR models [19], [31] we assume that a query q is generated by a probabilistic model based on a candidate result graph g. Given a query $q = q_1 q_2 \cdots q_n$ and a candidate answer $g = g_1 g_2 \cdots g_n$, where each q_i is a fact template and each g_i is a fact, we want to estimate the conditional probability $P(g|q)$, i.e. the probability that g generated the observed q [31].

After applying Bayes formula and dropping a graph-independent constant (since we are only interested in ranking graphs), we have $P(g|q) \sim P(q|g)P(g)$ where $P(g)$ can reflect a prior belief that g is relevant to any query. $P(q|g)$ is the query likelihood given the graph g which captures how well the graph fits the particular query q. In our setting we assume $P(g)$ to be uniform and thus we are interested in computing $P(q|g)$. In the spirit of IR models [31], we assume probabilistic independence between the query's fact templates which results in $P(q|g) = \prod_{i=1}^{n} P(q_i|g)$. Our intuition behind the independence assumption is based on the independent extraction of facts in the construction phase of NAGA's knowledge base (see [30]). Furthermore, the independence assumption between data items is used as a standard technique to avoid high computational complexity in high-dimensional data settings ("the curse of dimensionality"). A well known example for such a setting is the vector space model for document retrieval where each document is represented as a vector and each dimension corresponds to a separate term. Analogously, in our setting the dimensions would be spanned by our facts.

Next, we design a $tf \cdot idf$ style probabilistic mixture model for query fact templates. We follow classical IR literature [19] but develop a new scoring model suited for our setting. We define the likelihood of a query fact given an answer graph as a mixture of two distributions, $\tilde{P}(q_i|g)$ and $\tilde{P}(q_i)$ as follows:

$$P(q_i|g) = \alpha \cdot \tilde{P}(q_i|g) + (1-\alpha) \cdot \tilde{P}(q_i), 0 \leq \alpha \leq 1 \qquad (2)$$

$\tilde{P}(q_i|g)$ is the probability of drawing q_i randomly from an answer graph, $\tilde{P}(q_i)$ is the probability of drawing q_i randomly from the total knowledge graph and α is either automatically learned (via EM iterations [19]) or set to an empirically calibrated global value. [19], [31] show the connection between this style of probabilistic models and the popular $tf \cdot idf$ heuristics.

As mentioned before, we want to capture *confidence*, *informativeness*, and *compactness*. We first describe the confidence and informativeness components and then explain how our model automatically deals with compactness. We describe $\tilde{P}(q_i|g)$ by a mixture model which puts different weights on confidence and informativeness. This is close in spirit to linear interpolation models used for smoothing [31]. β is empirically calibrated as analyzed in our evaluation section.

$$\begin{aligned}\tilde{P}(q_i|g) &= \beta \cdot P_{conf}(q_i|g) + (1-\beta) \cdot P_{info}(q_i|g) \qquad (3)\\ & 0 \leq \beta \leq 1\end{aligned}$$

Note that the confidence and the informativeness are indeed independent criteria. For example, we can be very confident that Albert Einstein was both a physicist and a politician, but the former fact is more informative than the latter, because Einstein was a physicist to a larger extent than he was a politician.

1) Estimating Confidence: The maximum likelihood estimator for $P_{conf}(q_i|g)$ is:

$$P_{conf}(q_i|g) \;=\; \prod_{f \in match(q_i,g)} P(f \text{ holds}) \qquad (4)$$

where $P(f \text{ holds})$ is estimated by $c(f)$ (see Formula (1)). If q_i is labeled by a simple relation name, then $match(q_i, g)$ contains just one fact and $P_{conf}(q_i|g)$ is the confidence of that fact. If q_i is labeled with connect or with a regular expression over relations, then $match(q_i, g)$ contains the sequence of facts that together match q_i. The likelihood of that sequence being true is the product of the confidences of the single facts, assuming that the facts are independent.

2) Estimating Informativeness: The *informativeness* of a query template q_i given the answer graph g depends on the informativeness of each matching fact in g:

$$P_{info}(q_i|g) \;=\; \prod_{f \,\in\, match(q_i,g)} P_{finfo}(f|q_i) \qquad (5)$$

Note that the same fact f may have different informativeness values, depending on the query formulation. For example, the fact Bob_Unknown instanceOf physicist would be less informative if the query asked for (famous) physicists ($x instanceOf physicist), but could be very informative if the query asked about the occupation of Bob_Unknown (Bob_Unknown instanceOf $x). Thus the informativeness of the fact f depends on the unbound arguments of the query template q_i.

Let $f = (x, r, y)$ be an observation drawn from the joint distribution of three random variables X, R and Y. X and Y take values from the set of knowledge graph nodes and R takes values from the set of edges (e.g. relations). Given a query template $q_i = (x', r', y')$, if $f = (x, r, y)$ is a match for q_i, we define the informativeness of f as follows:

$$P_{finfo}(f|q_i) = \begin{cases} P(x|r,y), & \text{if } x' \text{ unbound in } q_i \\ P(y|r,x), & \text{if } y' \text{ unbound in } q_i \\ P(r|x,y), & \text{if } r' \text{ unbound in } q_i \\ P(x,y|r), & \text{if } x', y' \text{ unbound in } q_i \\ P(x,r|y), & \text{if } x', r' \text{ unbound in } q_i \\ P(r,y|x), & \text{if } r', y' \text{ unbound in } q_i \\ P(x,r,y), & \text{else} \end{cases} \quad (6)$$

We show how to estimate these probabilities by the example of $P(x|r, y)$. $P(x|r, y)$ can be written as follows:

$$P(x|r,y) = \frac{P(x,r,y)}{P(r,y)} = \frac{P(x,r,y)}{\sum_{x'} P(x',r,y)} \quad (7)$$

We estimate $P(x, r, y)$ using the number of witness pages for the fact (x, r, y) [2]:

$$P(x,r,y) \approx \frac{|W(x,r,y)|}{\sum_{x',r',y'} |W(x',r',y')|} \quad (8)$$

To see why this formulation captures the intuitive understanding of informativeness, consider some examples. Let q be the query $q=$ Albert_Einstein instanceOf $x, which consists of one fact template. Let f be a possible answer $f=$ Albert_Einstein instanceOf physicist. Here, the informativeness of f measures how often Einstein is mentioned as a physicist as compared to how often he is mentioned with some other instanceOf fact. Thus, $f=$ Albert_Einstein instanceOf physicist will rank higher than $f' =$ Albert_Einstein instanceOf politician. In this case, informativeness measures the *degree* to which Einstein was a physicist.

Now consider the query $q =$ $x instanceOf physicist and consider again the answer $f =$ Albert_Einstein instanceOf physicist. The informativeness of f will compute how often Einstein is mentioned as a physicist compared to how often other people are mentioned as physicists. Since Einstein is an important individual among the physicists, Albert_Einstein instanceOf physicist will rank higher than Bob_Unknown instanceOf physicist. In this case, informativeness measures the *importance* of Einstein in the world of physicists.

More examples could be: When asking for prizes that Einstein won, our informativeness will favor the prizes he is most known for. When asking for people born in some year, informativeness favors famous people. When asking for the relationship between two individuals, informativeness favors the most prominent relation between them.

For now the number of witnesses for each fact in our knowledge graph is not statistically significant, because our facts are extracted only from a limited number of Web-based corpora, and many facts appear only on one page. For this reason we approximated the $P(x, r, y)$ values by a heuristic. We transformed the facts into keyword queries and used a search engine to retrieve the number of pages in the Web that contain the corresponding keywords. For example, to estimate $P($Albert_Einstein|instanceOf, physicist$)$, we formulated the query "Albert Einstein" + "physicist" and retrieved the number of hits for this query. We retrieved the number of hits for the query "physicist" as well and estimated the probability as follows:

$$\begin{aligned} &P(Albert\_Einstein|instanceOf, physicist) && (9) \\ &\sim P(Albert\_Einstein|physicist) && (10) \\ &= \frac{P(Albert\_Einstein, physicist)}{P(physicist)} && (11) \\ &\sim \frac{\#hits(Albert\ Einstein\ physicist)}{\#hits(physicist)} && (12) \end{aligned}$$

Note that using the graph structure (e.g. in-degree of nodes) as an alternative for computing informativeness will lead to arbitrary ranking behavior, e.g. for the query Albert_Einstein instanceOf $x, the answer Albert_Einstein instanceOf politician will be ranked higher than the answer Albert_Einstein instanceOf physicist just because there are more instances of politicians worldwide than of physicists (see also comparison to BANKS scoring [6] in Section V).

In summary, confidence and informativeness are two complementary components of our model. The *confidence* expresses how certain we are about a specific fact – independent of the query and independent of how popular the fact is on the Web. The *informativeness* captures how useful the fact is for a given query. This depends also on how visible the fact is on the Web. In this spirit, our definition of informativeness differs from the information theoretic one, which would consider less frequent facts more informative. The latter is captured by our background model (see III-A.4). For the estimation of $P_{finfo}(f|q_i)$ we rely on asymmetric measures (see (6), (11), (12)), which reflect the position of variables in the fact templates. Therefore, symmetric information theoretic measures, such as PMI (pointwise mutual information), would not be an adequate choice for the estimation of $P_{finfo}(f|q_i)$.

3) Estimating Compactness: The *compactness* of answers is implicitly captured by their likelihood given the query. This is because the likelihood of an answer graph is the product over the probabilities of its component facts. Therefore, the more facts in an answer graph the lower its likelihood and thus its compactness.

For example, for the query Margaret_Thatcher connect Indra_Gandhi the answer graph stating that they are both prime-ministers, is more compact than the answer that they are both prime-ministers of English-speaking countries.

4) The Background Model: We turn to estimating $\tilde{P}(q_i)$, which plays the role of giving different weights to different fact templates in the query. This is similar in spirit to the idf-style weights for weighting different query terms in traditional LMs. For a single-term query the *idf* part would just be a constant shift or scaling, which does not influence the ranking.

[2] The witnesses could also be weighted by their authority, e.g. Page Rank.

Algorithm 1 Query Processing Algorithm

```
1:  Function: queryResults($Q$)
2:  Input: A query graph $Q = (V_Q, E_Q, L_{E_Q}, L_{V_Q}, U)$
3:  Output: A set of answer graphs
4:  normalize $Q$ into $Q' = (V_{Q'}, E_{Q'}, L_{E_{Q'}}, L_{V_{Q'}}, U)$
5:  return templateResults($Q', E_{Q'}$)
1:  Function: templateResults($C$,$E$)
2:  Input: A query graph $C = (V_C, E_C, L_{E_C}, L_{V_C}, U)$
3:  Input: A set of fact templates $E$
4:  Output: A set of answer graphs
5:  If $E = \emptyset$, return $\{C\}$
6:  $Results = \emptyset$
7:  Pick a fact template $e \in E$
8:  for all matches $e'$ of $e$ in the knowledge graph do
9:    $r_{e'}$=templateResults(($V_C$,$E_C - e + e'$,$L_{E_C}$,$L_{V_C}$,$U$),$E - e$)
10:   If $r_{e'} \neq \emptyset$, $Result = Result + r_{e'}$
11: end for
12: return $Results$
```

But for multi-term queries, the idf weights give different importance to different query terms. For example, consider the query with two fact templates $q1 =$ $y bornIn Ulm and $q2 =$ $y isA scientist. If matches to this query are only partial, e.g. answers in which only one of the fact templates is matched are allowed, then the more important template should get higher weight. Traditionally, the more important condition is the more specific one – the one that is expected to have fewer matches, i.e., higher idf. If there are many people born in Ulm, but there are only few scientists overall, this suggests giving a higher weight to $q2$. By counting edges of the form $x bornIn Ulm and $x isA scientist in the overall corpus (knowledge graph), we get corresponding frequency (an estimate for $\tilde{P}(q_i)$) and thus inverse frequency weights, in the idf spirit. This type of background model is heavily used in standard IR [19], [31].

IV. Query Processing

NAGA stores the facts of the knowledge graph into a database table with the schema *Facts*(ID, RELATION, ENTITY1, ENTITY2, CONFIDENCE). A high-level overview of NAGA's query processing algorithm is shown in **Algorithm 1**[3]. We first pre-process the given query into a normalized form (line 4, Function `queryResults`) by applying the following rewritings: First, because we allow users to use words for referring to entities, we add an additional edge labeled with means for each bound vertex, e.g. the query Einstein hasWonPrize $x becomes "Einstein" means $Einstein; $Einstein hasWonPrize $x. Second, we translate the pseudo-relation isA to its explicit form instanceOf subclassOf*, e.g. the query $x isA $y becomes $x instanceOf subclassOf* $y. This allows the user to ask for instances of classes without the need to know about regular expressions.

The main function of the query processing algorithm is `templateResults`. It is given a preprocessed query graph and a list of templates to be processed. Initially, the templates are edges of the query graph. Some edge is picked (line 7) and all possible matches of this edge in the query graph are identified. For each possible match, we construct a refined query graph by replacing the fact template by the match. Then, the function is called recursively with the refined query graph. Once no more query fact templates need to be processed, the refined query graph constitutes a result.

We identify matches for templates as follows. If the label of the edge in the fact template is a **simple relation** or a variable, we translate the template directly to an SQL statement. This applies to templates like "Einstein" means $z, "Einstein" $r Ulm, or $x invented $z, which can be translated into simple SELECT statements over the *Facts* table.

If the edge of the template is labeled with a **regular expression** over relations, we construct a (non-deterministic) automaton for the regular expression. We identify one vertex v_0 of the edge that is already bound. If v_0 is not the start vertex of the edge, but the target vertex, we invert the automaton. Now, we start a breadth-first-search in the knowledge graph from v_0. The visited vertices of the search are stored in a queue. To each vertex in the queue, we attach (1) a set of states of the automaton and (2) a predecessor vertex. Initially, the queue contains just v_0 with the initial states of the automaton and no predecessor node attached. Whenever we poll a vertex v from the queue, we examine the automaton states attached to v. For each state s, we find all possible vertices v' in the knowledge graph that occur in edges $e = (v, v')$ with $l(e) = r$, where r is the relation label of the regular expression that is being read by the automaton. To each such v', we attach v as a predecessor vertex and the successor states of s as its states. When one of the successor states of s is an exit state of the automaton, we obtain a matching path for the regular expression edge by following the predecessor vertices of v'. Then, v' is enqueued. This process continues until the queue is empty. This gives us a set of matching paths for the template.

In case the template is labeled with connect, we search a chain of facts from the first template argument to the second. We implement it by two breadth-first-searches, which start from the two vertices and grow until they meet. This process can deliver multiple paths between the arguments, if desired. This gives us a set of matching paths for the connect template.

We also incorporate some query optimizations: First, fact templates in which the edge as well as both vertices are not labeled by a variable are processed separately, so that they do not need to be computed in each recursive call. Second, we coalesce subsequent non-regular expression edges to one single SQL statement whenever possible. Furthermore, certain trivial relations (such as e.g. smallerThan for numbers or before and after for dates) are not stored in the database, but are computed at query time.

V. Evaluation

In this section we evaluate NAGA's search and ranking behavior. First, we discuss the influence of the model parameters on the ranking desiderata, i.e. *confidence*, *informativeness*, and *compactness*. Then, we present an extensive user study that compares NAGA's performance to the performance of Google,

[3] Given a set of edges E and a match e, we write $E + e$ for $E \cup \{e' : e'$ is an edge in $e\}$. Note that e may be a sequence of edges.

Yahoo! Answers, and START[4] [23]. Finally, we compare NAGA's scoring mechanism to the one of BANKS [6].

A. Influence of ranking parameters

As discussed in Section III, the parameter α can be used to give different weights to different fact templates of a query by means of the background model $\tilde{P}(q_i)$. For our study we focus on exact matches. Thus we set $\alpha = 1$.

On the other hand, the parameter β can be used to formulate a more flexible scoring, in which either confidence or informativeness is given a higher emphasis. For example, if we search for a drug that heals malaria, we would want to emphasize confidence more than informativeness, i.e. we would not be interested in famous drugs for malaria, but in drugs that have high associated confidence for healing the disease. If we want to find out new meanings associated with a word, we may emphasize the informativeness more than the confidence. This would promote information that appears in many possibly low confidence sources, e.g. revealing that the word Kleenex (which is a trademark) is used by many people with the meaning of tissues. For our experiments we set β to the balanced value 0.5 giving equal weight to informativeness and confidence.

B. User Study

a) Benchmarks: We evaluated NAGA on three sets of queries. Sample queries from each of these sets are shown in Table I.

Benchmark	Question with NAGA translation
TREC	When was Shakespeare born? Shakespeare bornInYear $x
	In what country is Luxor? Luxor locatedIn $x $x isA country
SphereSearch	In which movies did a governor act? $y isA governor $y actedIn $z $z isA movie
	What was discovered in the 20th century? $x discoveredInYear $y $y after 1900 $y before 2000
OWN	Who produced or directed the movie ”Around the World in 80 Days”? $x produced|directed Around_the_World_in_80_Days
	What do Albert Einstein and Niels Bohr have in common? Albert_Einstein connect Niels_Bohr

TABLE I
SAMPLE QUERIES

- TREC 2005 and TREC 2006 provide standard benchmarks for question answering systems. Out of this set, we determined the questions that can be expressed by the current set of NAGA relations. We obtained a set of 55 questions (**query set TREC**). Note that although NAGA knows the relations used in the questions, the knowledge graph does not necessarily have the data instances to answer them.
- The work on SphereSearch [15] provides a set of 50 natural language questions for the evaluation of a search engine. Again, we determined the 12 questions that can be expressed in NAGA relations (**query set SphereSearch**).
- Since, to the best of our knowledge, we are the first to utilize regular expressions over relations and relatedness queries, we had to provide these queries by ourselves. We constructed 18 corresponding natural language questions (**query set OWN**).

b) Competitors: Given that the established search and question answering (QA) systems use different corpora, data models, query languages and rankings, the evaluation becomes very difficult. Nevertheless, in our study we try to cover a broad spectrum of retrieval systems and techniques, by comparing ourselves to state-of-the-art systems: Google (search engine), Yahoo! Answers and START (QA systems). Furthermore, in order to have a homogeneous evaluation of NAGA's scoring mechanism we compare it to the one used by an established graph-based search engine, BANKS [6].

It is clear that these systems are considerably different. Google is designed to find Web pages, not to answer questions. Still, it is a robust competitor, because of its large amount of indexed Web pages. It is also tuned to answer specific types of questions (e.g. *When was Einstein born?*) directly by its built-in QA system. Yahoo! Answers has its own corpus of questions and corresponding answers (provided by humans). When given a question, it first matches it to a question in its corpus and retrieves the answer. START is an established QA system, which understands natural language questions and can give answers based on information gathered from the Web. BANKS performs keyword search over the graph-oriented representation of a database. The nodes of the graph represent tuples from database tables and the edges represent foreign-key relationships between tuples. The answers to a query are graphs containing the query keywords. To evaluate our scoring function explicitly, we compare the NAGA scoring mechanism to the one proposed for BANKS. For this purpose, we integrated the BANKS scoring function into the NAGA engine and compared it to NAGA's own scoring mechanism.

All the questions were posed to Google, Yahoo! Answers, START and NAGA (with NAGA scoring and BANKS scoring, respectively). While for Google, Yahoo! Answers and START the queries were posed in their original natural language form, for NAGA the queries were posed in their graph form (see Table 1). This type of comparison is influenced by several aspects: First, the results returned by a system in this evaluation depends on how precisely the questions can be formulated.

[4] http://start.csail.mit.edu/

Benchmark	#Q	#A	Measure	Google	Yahoo! Answers	START	BANKS scoring	NAGA
TREC	55	1098	NDCG	75.88% ± 6.28%	26.15% ± 6.46%	75.38% ± 5.31%	87.93% ± 3.95%	**92.75%** ± 3.11%
			P@1	67.81% ± 6.87%	17.20% ± 5.52%	73.23% ± 5.46%	69.54% ± 5.63%	**84.40%** ± 4.42%
SphereSearch	12	343	NDCG	38.22% ± 11.22%	17.20% ± 8.54%	2.87% ± 2.87%	88.82% ± 6.80%	**91.01%** ± 6.07%
			P@1	19.38% ± 8.98%	6.15% ± 5.01%	2.87% ± 2.87%	84.28% ± 8.00%	**84.94%** ± 7.84%
OWN	18	418	NDCG	54.09% ± 11.29%	17.98% ± 8.54%	13.35% ± 6.92%	85.59% ± 6.75%	**91.33%** ± 5.28%
			P@1	27.95% ± 10.10%	6.57% ± 5.13%	13.57% ± 6.97%	76.54% ± 8.25%	**86.56%** ± 6.54%

#Q – Number of questions
#A – Total number of assessments for all questions

TABLE II
RESULTS

Second, it depends on the size of the knowledge base that the system uses. Third, the comparison measures the quality of the ranking of a system. Clearly, NAGA has an advantage over Google, Yahoo! Answers and START, because of its graph-based query language. At the same time, Google and Yahoo! Answers have a massive advantage over NAGA, because they are commercially operated systems that can search the whole Web (Google) or have a huge corpus of several million predefined questions (Yahoo! Answers). START is explicitly designed to answer questions.

c) Measurements: For each question, the top-ten results of all systems were shown to human judges. On average, every result was assessed by 20 human judges. For each result of each system, the judges had to decide on a scale from 2 to 0, whether the result is highly relevant (2), correct but less relevant (1), or irrelevant (0).

NAGA answers queries by finding matches in the knowledge graph. For example, for a query such as Albert Einstein bornInYear $x, NAGA returns only the result Albert Einstein bornInYear 1879. Hence the direct comparison to the other systems in terms of the well known precision-at-top-10 (P@10) measure would be misleading. Therefore we chose a measure that is not dependent on the number of results returned by the system for a given query, and which additionally can exploit the rank and the weight of relevant results in the result list. The *Normalized Discounted Cumulative Gain (NDCG)* was introduced by [21] and is intensively used in IR benchmarking (e.g. TREC). It computes the cumulative gain the user obtains by examining the retrieval results up to a fixed rank position. The NDCG rewards result lists in which highly relevant results are ranked higher than marginally relevant ones. We average the NDCG for one query over all user evaluations for that query and average these values over all queries.

Furthermore, we provide the *precision at one* (P@1) in order to measure how satisfied the user was on average with the first answer of the search engine. P@1 is the number of times that a search engine provided a relevant result in the first position of the ranking, weighted by the relevance score (0 to 2) and normalized. To be sure that our findings are statistically significant, we compute the Wilson confidence interval for the estimates of NDCG and P@1. We report confidence intervals for a confidence level of $\alpha = 95\%$.

d) Results: Table II shows the results of our evaluation. For the TREC query set, Google performs very well. The first hit in its result ranking is a satisfactory answer. The reason for this is that the TREC questions are mostly of a very basic nature (see Table 1) and Google can answer a major part of them directly by its highly precise built-in question answering system. In contrast, Yahoo! Answers performs less well. Very often, it retrieves answers to questions that have only the stop-words in common with the question posed. In many cases, it does not deliver an answer at all. START performs much better than Yahoo! Answers. Whenever it has the appropriate data in its knowledge base, its answers are highly satisfactory.

The SphereSearch questions are of a more sophisticated nature. They ask for a non-trivial combination of different pieces of information. Consequently, both Google and Yahoo! Answers perform worse than for the TREC questions. START performs poorly here, often because it does not understand the question (it tries to parse proper names as English words) and often because it does not know the answer. NAGA, in contrast, excels on these questions, because it makes full use of its graph-based query language.

For OWN queries, Google again performs quite well. This is because the questions mostly ask for a broad relationship between two individuals. Google can answer these questions by retrieving Web documents that contain the two keywords. In most cases, these answers were satisfactory. Yahoo! Answers had again difficulties. START could not answer questions that ask for the broad relationship between two entities (no matter how we phrased the question) and therefore often failed. NAGA delivers good results for the majority of questions and clearly outperforms Google.

As shown in Table II (columns 8, 9) NAGA's scoring mechanism outperforms the scoring mechanism of BANKS. The BANKS scoring function is given by an interpolation of the average in-degree of nodes and the average edge weights in a result graph (see [6]). Hence, it relies only on the graph structure, which is not enough to capture informativeness. When asked for (famous) politicians, the BANKS scoring returns *Albert Einstein* as the first result. For the query Albert_Einstein isA $x the BANKS scoring returns *person* as the first result. This is because of the high in-degree of the nodes representing the entities *Albert Einstein* and *person* in the knowledge graph. NAGA's scoring mechanism, instead, is a powerful combination of graph structure properties and fact witness estimations. When asked for (famous) politicians, NAGA's scoring returns *Barack Obama* as the first result, while for the query Albert_Einstein isA $x, the first answer is *physicist*.

VI. Related Work

The most prominent works along similar lines are probably Libra [27], EntitySearch [9], and ExDBMS [7], all of which also operate on relations extracted from Web data. Libra focuses on entities and their attributes, using a novel record-level language model. However, it does not address general relations between different entities, and its query model is keyword-centric. EntitySearch [9] facilitates search that can combine keywords and structured attributes in a convenient manner, and it has an elaborate ranking model for result entities. However, it does not address typed relations between entities and SPARQL-style path expressions on knowledge graphs, and its ranking model is very different from ours. ExDBMS [7] uses powerful IE tools to capture entities and typed relations, and its query model supports this full suite as well. It uses a probabilistic form of Datalog for search [11]. In contrast, NAGA uses a graph-based search paradigm that is more expressive by supporting regular expressions on paths and a very general form of relatedness queries, with joins along edges as a special case. Furthermore, NAGA extends the above approaches by considering informativeness and confidence levels of imperfect extraction, at query time, by means of an elaborate ranking method for complex queries based on principles of statistical language models, the latter being one of the cornerstones of modern information retrieval (IR) for text documents [25], [31].

Our query language is akin to the RDF query language SPARQL and XML query languages such as XPath or XQuery. However, all these languages disregard the issue of uncertainty. More related to our work is the research on XML IR for ranked retrieval (see [3] and the references given there). This line of work, however, does not consider graph structures that reflect Web connectivity or arbitrary relations that could be viewed as typed edges in a graph. SPARQL-style languages are geared for graphs (see, e.g., [4] for recent, powerful models), but do not consider uncertainty and treat ranking as a second-class citizen.

Deep-Web search, vertical search and entity search on the Web, and semantic desktop search [8], [9], [12], [13], [27], [26] enhance keyword-based querying by typed attributes, but none of these approaches is sufficiently complete for effectively searching a richly structured knowledge base.

Finally, there is prior work on *graph-oriented text search* in various forms. Schema-oblivious keyword search in relational databases operates on a graph that is created by the foreign-key relationships in the database. BANKS [6], DBXplorer [2], and DISCOVER [20] are the most prominent systems of this kind. More recent work along these lines has focused on efficiency issues [5], [18], [22], [24], [28] and did not explore richer functionality. These kinds of data graphs can be generalized into networks of entities and relationships, and similar graph structures also arise when considering XML data with XLinks and other cross-references within and across document boundaries [16], [10]. However, these methods operate on text nodes in a graph, but without using explicit relation types in advanced queries. In contrast, NAGA has a notion of typed edges that correspond to binary relations between entities, and can utilize this additional knowledge for more precise querying along with a principled and yet general ranking model that can be applied to any kind of labeled, weighted data graph.

VII. Conclusions

In this paper, we presented the NAGA search engine, which facilitates advanced querying for knowledge rather than merely retrieving Web pages. NAGA's large knowledge base, organized as a directed graph, consists of millions of facts extracted from Web-based corpora. We introduced a graph-based query language with several distinctive features. We proposed a novel scoring mechanism based on generative language models, incorporating the notions of confidence, informativeness, and compactness in a principled manner. We compared our system to state-of-the-art retrieval systems by conducting a comprehensive user study on a variety of simple and complex queries. The results demonstrated that NAGA returns answers which are superior in quality to all competitors.

Our future work will aim to further increase the scale and scope of our knowledge base by incorporating more sources. A second aspect is query-processing efficiency. Although, for almost all queries, the first result is returned in less then a second, the efficiency for relatedness queries should be improved. We plan to adapt recent algorithms for keyword search in relational graphs for this purpose. NAGA is available online at `http://www.mpii.mpg.de/~kasneci/naga`.

References

[1] E. Agichtein, S. Sarawagi. Scalable information extraction and integration. Tutorial. KDD, 2006.

[2] S. Agrawal, S. Chaudhuri, G. Das. DBXplorer: A system for keyword-based search over relational databases. ICDE, 2002.

[3] S. Amer-Yahia, J. Shanmugasundaram. XML full-text search: Challenges and opportunities. Tutorial. VLDB, 2005.

[4] K. Anyanwu, A. Maduko, A. Sheth. SPAR2QL: Towards support for subgraph extraction queries in rdf databases. WWW, 2007.

[5] B. Ding, J. Yu, S. Wang, L. Qin, X. Zhang, X. Lin Finding top-k min-cost connected trees in databases. ICDE, 2007.

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

[7] M. Cafarella, C. Re, D. Suciu, O. Etzioni. Structured querying of web text data: A technical challenge. CIDR, 2007.

[8] S. Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. WWW, 2007.

[9] T. Cheng, K. C.-C. Chang. Entity search engine: Towards agile best-effort information integration over the web. CIDR, 2007.

[10] S. Cohen, Y. Kanza, B. Kimelfeld, Y. Sagiv. Interconnection semantics for keyword search in XML. CIKM, 2005.

[11] N. Dalvi, D. Suciu. Efficient query evaluation on probabilistic databases. VLDB, 2004.

[12] J.-P. Dittrich, M. A. V. Salles. iDM: A unified and versatile data model for personal dataspace management. VLDB, 2006.

[13] X. Dong, A. Y. Halevy. A platform for personal information management and integration. CIDR, 2005.

[14] C. Fellbaum, editor. WordNet: An Electronic Lexical Database. MIT Press, 1998.

[15] J. Graupmann. The SphereSearch Engine for Graph-based Search on heterogeneous semi-structured data. PhD thesis, Universität des Saarlandes, 2006.

[16] J. Graupmann, R. Schenkel, G. Weikum. The Spheresearch engine for unified ranked retrieval of heterogeneous XML and web documents. VLDB, 2005.

[17] J. Han, X. Yan, P. Yu. Mining and searching graphs and structures, tutorial. KDD, 2006.

[18] H. He, H. Wang, J. Yang, P. Yu. BLINKS: Ranked keyword searches on graphs. SIGMOD, 2007.

[19] D. Hiemstra. A probabilistic justification for using tf.idf term weighting in information retrieval. Int. J. on Digital Libraries 3(2), 2000.

[20] V. Hristidis, Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. VLDB, 2002.

[21] K. Jarvelin, J. Kekalainen. IR evaluation methods for retrieving highly relevant documents. ACM Press, 2000.

[22] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, H. Karambelkar. Bidirectional expansion for keyword search on graph databases. VLDB, 2005.

[23] B. Katz, G. Marton, G. Borchardt, A. Brownell, S. Felshin, D. Loreto, J. Rosenberg, B. Lu, F. Mora, S. Stiller, O. Uzuner, A. Wilcox. External Knowledge Sources for Question Answering. TREC, 2005.

[24] B. Kimelfeld, Y. Sagiv. Finding and approximating top-k answers in keyword proximity search. PODS, 2006.

[25] X. Liu and W. Croft. Statistical language modeling for information retrieval. Annual Review of Information Science and Technology 39, 2004.

[26] J. Madhavan, S. Cohen, X. Dong, A. Halevy, S. Jeffery, D. Ko, C. Yu. Navigating the seas of structured web data. CIDR, 2007.

[27] Z. Nie, Y. Ma, S. Shi, J. Wen, W. Ma Web object retrieval. WWW, 2007.

[28] M. Sayyadan, H. LeKhac, A. Doan, L. Gravano. Efficient keyword search across heterogeneous relational databases. ICDE, 2007.

[29] F. M. Suchanek, G. Ifrim, G. Weikum. Combining Linguistic and Statistical Analysis to Extract Relations from Web Documents. KDD, 2006.

[30] F. M. Suchanek, G. Kasneci, G. Weikum. Yago: A Core of Semantic Knowledge. WWW, 2007.

[31] C. Zhai, J. Lafferty. A risk minimization framework for information retrieval. Information Proc. and Management 42, 2006.