Exploiting Relevance Feedback in Knowledge Graph Search

Yu Su[1], Shengqi Yang[1],Huan Sun[1], Mudhakar Srivatsa[2], Sue Kase[3], Michelle Vanni[3], and Xifeng Yan[1]
{ysu, sqyang, huansun, xyan}@cs.ucsb.edu,
msrivats@us.ibm.com,
{sue.e.kase.civ, michelle.t.vanni.civ}@mail.mil
[1]University of California, Santa Barbara, [2]IBM Research, [3]U.S. Army Research Lab

ABSTRACT

The big data era is witnessing a prevalent shift of data from homogeneous to heterogeneous, from isolated to linked. Exemplar outcomes of this shift are a wide range of graph data such as information, social, and knowledge graphs. The unique characteristics of graph data are challenging traditional search techniques like SQL and keyword search. Graph query is emerging as a promising complementary search form. In this paper, we study how to improve graph query by relevance feedback. Specifically, we focus on knowledge graph query, and formulate the *graph relevance feedback* (GRF) problem. We propose a general GRF framework that is able to (1) tune the original ranking function based on user feedback and (2) further enrich the query itself by mining new features from user feedback. As a consequence, a query-specific ranking function is generated, which is better aligned with the user search intent. Given a newly learned ranking function based on user feedback, we further investigate whether we shall re-rank the existing answers, or choose to search from scratch. We propose a strategy to train a binary classifier to predict which action will be more beneficial for a given query. The GRF framework is applied to searching DBpedia with graph queries derived from YAGO and Wikipedia. Experiment results show that GRF can improve the mean average precision by 80% to 100%.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

Keywords

Knowledge Graph; Graph Query; Relevance Feedback

1. INTRODUCTION

Querying graph data such as information, social, and knowledge graphs is always a challenging task. On the one hand, due to their complex schemas and varying information descriptions, it is extremely hard for users to formulate structured queries like SPARQL without spending hours digesting the schema [24]. On the other hand, unstructured search techniques like keyword search are not expressive enough to explore the structure of data to the maximum extent.

Graph querying [37, 17, 24], where users formulate information needs as graph patterns based on their own knowledge and vocabulary and look for matches in a graph, was recently under active investigation as a promising search technique for graph data. We focus on graph query over knowledge graphs in this work. Knowledge graphs contain a wealth of valuable information, with nodes representing entities and edges representing various relations between entities. Recent years have witnessed the blossom of large-scale knowledge graphs, such as DBpedia [18], Freebase [8], Google's Knowledge Graph [1], and YAGO [31].

There are other query forms for knowledge graphs that can be converted to graph query, such as logic query [7], natural language query [6], and exemplary query [15, 22]. Graph query can also serve as an intermediate query form to support question answering [42]. Figure 1 shows a graph query "Find a Toronto professor at age of 70 who joined Google recently," and its possible match. We put a question mark "?" on the node we are searching for an answer.

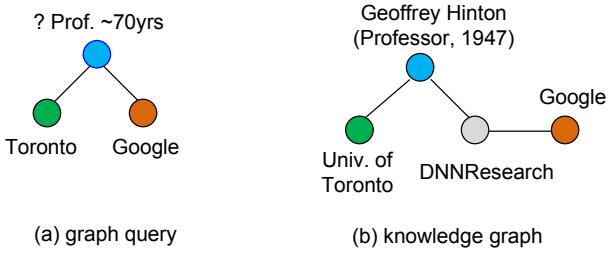


Figure 1: A query and its match.

Since the query is formulated based on the user's own vocabulary, there come vagueness and ambiguity. If $\sim 70yrs$ means one's age, it implies the person was *born around 1945*. *Toronto* is possibly a match for *University of Toronto*. Furthermore, *Google* and *Geoffrey Hinton* are not directly connected in the knowledge graph, but by a third node, *DNNresearch*, built on the fact that *Google* acquired *DNNresearch* founded by *Geoffrey Hinton*. For this query, Prof. Hinton shall be returned as a candidate answer.

KDD'15, August 11-14, 2015, Sydney, NSW, Australia.
© 2015 ACM. ISBN 978-1-4503-3664-2/15/08 ...$15.00.
DOI: http://dx.doi.org/10.1145/2783258.2783320.

The problem of answering a knowledge graph query is to find the matches of the query in a knowledge graph and sort them with respect to a ranking function. The goal is to return the best set of answers a user is looking for, a central problem of accessing knowledge graphs. The usage of graph query is not limited to knowledge graphs. It is also evident in e.g., social networks [33], cyber security [11] and data centers [41]. The task of answering graph queries is very challenging for two reasons. First, the vocabulary of different users can vary dramatically. According to a prominent study on the human vocabulary problem [13], about 80-90% of the times two persons will give different representations when they are asked to name the same concept. Second, the underlying data graphs are highly heterogeneous. For example, Google Knowledge Graph [1] contains over 35 thousand different types of edges, while YAGO [31] has over 350 thousand different types of nodes. Such high heterogeneity results in miscellaneous query interpretations.

A graph query technique usually crafts a generic ranking function to answer all queries [37]. However, given the high variety in query formulation, such functions are usually suboptimal for individual queries, resulting in poor answer quality. A natural solution to alleviate this situation is to develop *query-specific* ranking functions, i.e., specifically tailored for each query. To achieve that, query ambiguity and vagueness need to be correctly resolved, which in turn requires new information in addition to the query itself. Relevance feedback (RF), i.e., users indicating the (ir)relevance of a handful of answers, is one promising source. It is user-friendly, easy to get. Relevance feedback has been studied extensively and proven to be effective in document retrieval [29, 9] and image retrieval [40]. However, despite its naturalness, it has not yet been studied for graph query. Due to the unique characteristics of graph query, traditional relevance feedback methods are not directly applicable. In this paper, we define and study the *graph relevance feedback* (GRF) problem, which aims to achieve query-specific search via relevance feedback, and thus improve the search quality.

Our GRF framework works towards query-specific search from two directions: (1) Tune the original ranking function based on the preference evidence mined from user feedback, and (2) enrich the original query with new information discovered from user feedback, that is, the information the user might have in mind but did not explicitly specify when formulating the query. For the former, we propose *Query-Specific Tuning*, which uses the original ranking function as prior and adaptively tunes it according to user feedback. For the latter, we propose algorithms to infer more information from user feedback: (1) *Type Inference* which infers the expected answer type of query nodes, and (2) *Context Inference* which infers the context of query nodes. Putting all these together, we produce a query-specific ranking function, and show that searching with the new ranking function can significantly improve answer quality.

While the primary goal of GRF is to improve answer quality, there is an accompanying efficiency issue. Given a new ranking function, the existing relevance feedback methods in IR usually re-rank all of the documents from scratch. However, it could be costly for knowledge graph queries as the number of potential matches could be huge. GRF has a second option: Re-rank the top-k answers in the original answer list. Certainly, this option might affect answer quality as the best answers might not show up in the original answer list due to the change of the ranking function. Therefore, GRF is faced with a decision: In what situation, it has to search the answers from scratch in order to assure answer quality? In this work, we propose strategies to train a binary classifier, which can help control the trade-off between answer quality and query response time.

Contributions To the best of our knowledge, this work is the first formal attempt to investigate relevance feedback in graph querying. We summarize the contributions as follows:

1. The relevance feedback problem was formulated for querying knowledge graphs.
2. We developed techniques to infer three types of implicit information hidden in user feedback: relative importance of node/edge mappings, type and context similarity. We then put all the implicit information together to produce a query-specific ranking function that is better aligned with the search intent of a query.
3. We identified the runtime-quality trade-off in answering graph queries with a newly learned ranking function: Re-rank or search from scratch. A mechanism based on binary classification was proposed to control the trade-off according to the preference of real applications.

We evaluated the GRF framework on answering graph queries in real-life knowledge graphs. Experiment results show that it can improve the precision of a state-of-the-art graph querying technique by 80% to 100%, and meanwhile make a good trade-off between quality and runtime.

2. PRELIMINARIES

A knowledge graph, $G = (V, E)$, is a labeled graph with nodes representing entities such as *Barack Obama* and edges representing various relations between entities, e.g., *Barack Obama* isPresidentOf *United States*. Each entity is associated with a set of types/classes, and the classes form a class hierarchy via the subClass relation. A graph query is a labeled graph $Q = (V_Q, E_Q)$. The labels on nodes and edges are provided from a user's own vocabulary, so that she does not need to have intimate knowledge of the schema in order to formulate a query.

2.1 Querying Knowledge Graphs

Search techniques for knowledge graphs vary across a spectrum of expressivity and user-friendliness. At one end are the structured query languages such as SPARQL [3], which are very expressive but require users to have a fairly good understanding of the underlying data schema. Due to the ever-growing heterogeneity of knowledge graphs, users often find themselves facing the information overload problem [24], i.e., the data schema is too complex to grasp. Lying at the other end are unstructured search techniques like keyword search, which are easy to use but can not express structural constraints in queries. Graph querying emerges in the middle of the spectrum: On the one hand, users can express their belief or constraints via the structure of the query graph. On the other hand, users formulate queries using their own knowledge and vocabulary, and thus can stay agnostic about the complex data schema. In this work, we will target *graph pattern matching* (graph query for short).

The accompanying query vagueness and ambiguity make effective answer ranking a first-class citizen in graph querying: the most relevant answers should be shown to the user first. Existing graph querying techniques have crafted various kinds of ranking functions. For example, [24] defines a ranking function considering both the syntactic similarity and the semantic coherence between queries and matches, while SLQ [37] employs a conditional random field (CRF) model to learn and estimate the probability that a match is relevant to a query. In general, one can decompose a ranking function into three steps: extracting a set of features to characterize each match, appropriately weighting each feature, and aggregating the weighted features to generate a final relevance score for each match. Without loss of generality, we denote a ranking function as $F(\phi(Q)|Q, \boldsymbol{\theta})$, which evaluates the relevance of a match $\phi(Q)$ to a query Q based on a few features and their corresponding weights $\boldsymbol{\theta}$.

We introduce the ranking function of SLQ to give a concrete example. To measure the relevance of a match $\phi(Q)$, SLQ relies on a set of transformations $\{f_i\}$ that matches a query node (edge) to a set of candidate entities (relations) in G. For example, in Figure 1, the abbreviation transformation could match *Prof.* to *Professor*, while the topology transformation could identify the path between *Geoffrey Hinton* and *Google* as an edge match. A set of weights $\boldsymbol{\theta} = \{\alpha_i, \beta_i\}$ is assigned to the transformations in a way that more selective transformations are assigned with higher weights. Given a query node $v \in V_Q$ and a candidate entity $\phi(v) \in V$, the node match score is defined as a weighted sum of the transformations:

$$F_V(v, \phi(v)) = \sum_i \alpha_i \cdot f_i(v, \phi(v)). \quad (1)$$

The edge match score is defined similarly:

$$F_E(e, \phi(e)) = \sum_i \beta_i \cdot f_i(e, \phi(e)), \quad (2)$$

where $e \in E_Q$ and $\phi(e)$ is an edge match in G which could be either an edge or a path.

SLQ employs a CRF-based probabilistic ranking function which aggregates the node and edge match scores to estimate the conditional probability of a match $\phi(Q)$ being relevant to Q under a given $\boldsymbol{\theta}$:

$$P(\phi(Q)|Q, \boldsymbol{\theta}) = \frac{1}{Z} \exp\Big(\sum_{v \in V_Q} F_V(v, \phi(v)) + \sum_{e \in E_Q} F_E(e, \phi(e))\Big), \quad (3)$$

where Z is a normalization factor. $F(\phi(Q)|Q, \boldsymbol{\theta})$ can be defined by $\log P(\phi(Q)|Q, \boldsymbol{\theta})$. SLQ is a specific example of a feature-based graph matching, where each f_i is a feature measure and α_i, β_i are the weight (importance) of this feature in the final ranking function.

2.2 Relevance Feedback

Despite the efforts researchers have made to craft a universally good ranking function, such a generic ranking function is often sub-optimal. For example, if a user is interested in the annual yield of apples (fruit), she will most likely get overwhelmed by results about Apple, the computer company and the yield of iPhones or iPads, unless she is able to articulate her information need very clearly to the query engine, which usually implies a tedious and frustrating trial-and-error procedure. A natural idea to improve search effectiveness is to somehow acquire more query-specific information for query disambiguation, and learn a query-specific ranking function. Relevance feedback is a user-friendly manner to provide additional information to guide a query engine to return better results.

Definition 1 (Graph Relevance Feedback) Given a query Q and a knowledge graph G, a ranking function F, a set of relevant (positive) matches $\mathcal{M}^+$, and a set of non-relevant (negative) matches $\mathcal{M}^-$, graph relevance feedback works to find a query-specific ranking function $\tilde{F}$ for Q based on the user feedback, such that other relevant matches will be ranked higher by $\tilde{F}$ than by F.

Note that we do not constrain the way for acquiring user feedback. It could be explicit feedback (user manually judging matches), implicit feedback (relevance information inferred from user behavior such as clicks), or even pseudo feedback (blindly assuming all top ranked matches from an initial search are relevant).

3. A GENERAL GRF FRAMEWORK

we provide an overview of a general framework to approach the GRF problem in this section. It has three key components: Query-specific Tuning, Type Inference, and Context Inference.

Query-specific Tuning The parameters $\boldsymbol{\theta}$ in a ranking function $F(\phi(Q)|Q, \boldsymbol{\theta})$ usually represent feature importance in a general sense. That is, how important each feature is when no additional query-specific information is available. However, for different users and queries, feature preferences differ. Therefore, we propose Query-specific Tuning to learn query-specific parameters $\boldsymbol{\theta}^*$ from user feedback, and therefore tune the generic ranking function to be better aligned with the query intent. We design a feature re-weighting mechanism and employ regularization to prevent overfitting to the user feedback (Section 4).

In addition to tuning the original ranking function, we also enrich the original query with additional discriminative information inferred from user feedback. When a user was formulating a query, there might be more information she had in mind but did not state explicitly. Back to Figure 1, by "Toronto", the user could mean *Toronto city* or *University of Toronto*. While she had more information in her mind, such as what type of things she was referring to (a city or a university), there is little chance for a graph query engine to infer such implicit information by solely looking at the keyword "Toronto". Adding this information back to the query may greatly improve answer quality. User feedback provides the possibility to reveal such information. Specifically, we propose algorithms to infer two types of implicit information from user feedback: entity type and entity context.

Type Inference It is observed that the relevant entities of a query node usually belong to the same, or at least similar, classes. Therefore, the type information of positive feedback could shed light on the implicit types of query nodes. For example, if a user marks *Michael Phelps* as relevant, it is more likely she is looking for some *persons*, not *places* or *films*. If we have a fine-grained ontology, we can further

infer that the user might be looking for *athletes* or even *Olympic athletes*. To quantify the implicit type information of a query node, we gather its corresponding entities in the positive matches (positive entities), and compute a relevance score for each candidate entity by measuring how similar it is to the positive entities based on their types (Section 5).

Context Inference When a user was formulating a query, she had a specific context in mind about the interested information. User feedback could also help infer such context. In document search, contexts are the words in the same sentence or paragraph with the matched keywords, while in knowledge graphs, the entities having a direct relation with the matched entities become their context. For example, *Toronto city* is the birthplace of many persons, the home to many companies, etc., while *University of Toronto* is the employer of many professors, in affiliation with many research institutes, and so on. In other words, an entity's context comprises of the entities adjacent to it in the knowledge graph. We employ the type distribution in an entity's context as a quantitative proxy. Similar to Type Inference, a relevance score is defined for each candidate entity by measuring the context similarity between the candidate entity and the positive entities (Section 6).

4. QUERY-SPECIFIC TUNING

Our GRF framework first tunes the original ranking function $F(\phi(Q)|Q,\boldsymbol{\theta})$ using the query-specific information provided by the user feedback. As we have discussed, the original parameter $\boldsymbol{\theta}$ reflects feature importance in a general sense. However, different users may prefer to formulate the same information need in different ways; and from time to time, a user's own preferences may also change. Therefore, each query brings its own view of feature importance $\boldsymbol{\theta}^*$, and user feedback provides us with the evidence to learn it.

Intuitively, we shall find $\boldsymbol{\theta}^*$ that maximizes the scores of the positive matches while minimizing the scores of the negative matches:

$$g(\boldsymbol{\theta}^*) = \frac{\sum_{\phi(Q)\in\mathcal{M}^+} F(\phi(Q)|Q,\boldsymbol{\theta}^*)}{|\mathcal{M}^+|} - \frac{\sum_{\phi(Q)\in\mathcal{M}^-} F(\phi(Q)|Q,\boldsymbol{\theta}^*)}{|\mathcal{M}^-|}. \quad (4)$$

However, a problem of Eq (4) is that it does not respect the original model $\boldsymbol{\theta}$, which contains valuable information that should not be ignored. In other words, $\boldsymbol{\theta}$ reflects which features are more important when we have no additional query-specific information. A better strategy is then to start from the original (query-independent) $\boldsymbol{\theta}$, and adjust it based on the (query-specific) evidence provided by the feedback. To achieve that, we add a regularization term to Eq (4).

$$g_\lambda(\boldsymbol{\theta}^*) = (1-\lambda)g(\boldsymbol{\theta}^*) + \lambda R(\boldsymbol{\theta},\boldsymbol{\theta}^*), 0 < \lambda \leq 1. \quad (5)$$

The regularization term R is a function of $\boldsymbol{\theta}$ and $\boldsymbol{\theta}^*$. An example is $-\|\boldsymbol{\theta}^*-\boldsymbol{\theta}\|_2^2$ if both $\boldsymbol{\theta}$ and $\boldsymbol{\theta}^*$ are vectors. It is designed to penalize $\boldsymbol{\theta}^*$ that deviates too far from $\boldsymbol{\theta}$. The balance parameter λ controls the trade-off between the query-independent information and the query-specific information: A large λ implies high sensitivity to the change of $\boldsymbol{\theta}^*$, which means that we highly respect the original parameters $\boldsymbol{\theta}$ and do not allow $\boldsymbol{\theta}^*$ to deviate far from it. On the other hand, when λ is small, we put more weight on the user feedback, and thus give more freedom for $\boldsymbol{\theta}^*$ to be adapted according to the feedback information. One can either manually set λ or learn it from training instances. The formal definition of query-specific tuning is given as follows:

Definition 2 (Query-specific Tuning) Given a query Q and the corresponding user feedback, a ranking function $F(\phi(Q)|Q,\boldsymbol{\theta})$, and a λ, the query-specific tuning algorithm maximizes $g_\lambda(\boldsymbol{\theta}^*)$ to find the optimal $\boldsymbol{\theta}^*$ for the query, and outputs a query-specific ranking function $F(\phi(Q)|Q,\boldsymbol{\theta}^*)$.

The convergence and complexity of the tuning algorithm depends on the mathematical form of the ranking function F and the regularization term R. One good practice is to carefully select them such that it becomes a convex optimization problem, for which efficient solutions exist.

Query-specific Tuning for SLQ To give a concrete example, we now describe how to apply query-specific tuning to SLQ. The ranking function of SLQ is given in Eq (3). We choose the following empirical form for ease of optimization:

$$F(\phi(Q)|Q,\boldsymbol{\theta}) = \sum_{v\in V_Q} F_V(v,\phi(v)) + \sum_{e\in E_Q} F_E(e,\phi(e)), \quad (6)$$

and L_2 regularization:

$$R(\boldsymbol{\theta},\boldsymbol{\theta}^*) = -\|\boldsymbol{\theta}^*-\boldsymbol{\theta}\|_2^2. \quad (7)$$

To get the final objective function, we substitute Eq (6) into Eq (4), and then substitute the result as well as Eq (7) into Eq (5). One can easily prove that the outcome is a convex function. The domain of $\boldsymbol{\theta}^*$ is the Euclidean space which is apparently a convex set. Therefore, we end up with a convex optimization problem, which can be efficiently solved by plenty of optimization tools. We use the open-source library JOptimizer [2] to do the optimization.

5. TYPE INFERENCE

Query-specific search can also be achieved by enriching the query itself with additional information. Types are one kind of such information. Types are very discriminative. For the query "Toronto", just knowing that the user is referring to a university could already filter out a lot of non-relevant answers, such as *Toronto city*, *Toronto Raptors* and *Chinatown (Toronto)*. However, if users do not explicitly put type information in queries, graph query engines are agnostic to it. Fortunately, user feedback, especially positive feedback, can help infer such implicit information. In this section, we propose a type inference algorithm to infer the type of entities from positive feedback.

In comparison with traditional relevance feedback in information retrieval, a significant advantage of knowledge graph for this task is that the type information is sometimes available. The *ontology* of a knowledge graph is where the type information resides. An ontology contains a set of classes (types) such as *person* and *place*, and these classes form a class hierarchy via the *subClass* relation. Each entity could be an instance of multiple classes, and therefore is associated with one or more types. For example, *Barack Obama* is a *politician*, *lawyer*, *writer*, etc., in Freebase.

For each query node, we have a set of positive entities from the positive matches. Given a candidate entity of the

query node, in order to examine whether its type(s) are what the query node expects, we can instead examine whether its type(s) are "similar" to the types of the positive entities. Computing semantic similarity against an ontology is a well-studied problem, and various semantic similarity measures have been proposed, each with its own pros and cons (see [14] for a comparison). We do not elaborate on the comparison of different semantic similarity measures, but choose the one based on information content [26] because it is intuitive and efficient to compute.

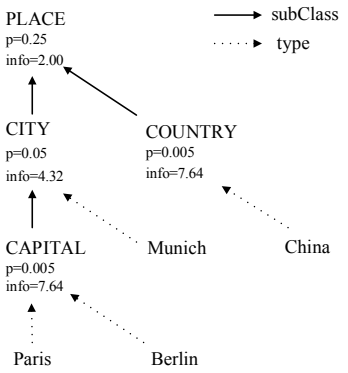


Figure 2: The computation of sim_{info}.

Information content measures how informative a class is. More formally, suppose there are in total N entities in the knowledge graph, among which n entities belong to a class c, the information content of c is then defined as $-\log p(c)$, where $p(c) = \frac{n}{N}$. In other words, a class bears more information content if it has fewer instances. The semantic similarity of two classes, c_1 and c_2, is the information content of their most informative superclass in the ontology:

$$sim(c_1, c_2) = max_{c \in S(c_1,c_2)}[-\log p(c)], \tag{8}$$

where $S(c_1, c_2)$ is the set of classes that subsume both c_1 and c_2. Furthermore, the semantic similarity of two entities ent_1 and ent_2 is defined as follows:

$$sim_{info}(ent_1, ent_2) = max_{c_1,c_2}[sim(c_1, c_2)], \tag{9}$$

where c_1 and c_2 range over the respective types of ent_1 and ent_2. In other words, it is the information content of the most informative class the two entities are both found in.

An example is shown in Figure 2. There are four classes, PLACE, CITY, COUNTRY, and CAPITAL forming a three-layer class hierarchy. The p attribute indicates the frequency of a class. The $info$ attribute indicates the information content of each class. Suppose p has the value shown in Figure 2. There are four entities, *Paris*, *Berlin*, *Munich* and *China*. Consider the semantic similarity of *Paris* to other entities. Intuitively, *Berlin* is the most similar entity to *Paris*, since they are both CAPITALs. *Munich* is less similar as a CITY. Lastly, *China* is the least similar entity to *Paris* because they are only common as PLACEs. The semantic similarity measure defined above can reflect this intuition. For example, the semantic similarity between *Paris* and *Berlin* $sim_{info}(Paris, Berlin) = 7.64$, is greater than $sim_{info}(Paris, Munich) = 4.32$.

After computing the pair-wise semantic similarity between a candidate entity and each positive entity, we define the type relevance score to measure the relevance of this candidate entity to the query node. We simply take the average (the range of sim_{info} is scaled to $[0, 1]$):

Definition 3 (Type Relevance Score) Given a query node v, a candidate entity ent, and a set of positive entities $\{ent_1, ..., ent_n\}$, the type relevance score of ent is defined as:

$$s_t(ent|v) = \frac{\sum_{i=1}^{n} sim_{info}(ent, ent_i)}{n}. \tag{10}$$

6. CONTEXT INFERENCE

People always bear a specific context in mind when they refer to something. Take "Toronto" for example. If the user means *Toronto city*, she might think it is the home to many companies' headquarters, the birthplace of many people, a beautiful place that has been written in many books, etc. On the other hand, if what she means is *University of Toronto*, then the context in her mind would be professors, students, departments, and so on. Since users rarely put such information into queries, we have no access to it. User feedback can help infer the context of a query and thus further disambiguate it.

We propose a context inference algorithm to infer the context of each query node from positive feedback. Quite different from the traditional relevance feedback in document search, where context often refers to the words in the same sentence or document with a matched keyword, the structure of knowledge graphs provide a more clear-cut definition of context: the context of an entity is the entities directly adjacent to it in the knowledge graph. It is possible to extend this definition by incorporating multiple-hop neighbors. To cope with the context in a quantitative manner, we use the type distribution in a context as its proxy. The context of *Toronto city* is now defined as the type distribution of entities that are linked to the node *Toronto city*:

Definition 4 (Contextual Type Distribution) Suppose there are L classes, $\{c_1, ..., c_L\}$, in the ontology, and n_i is the number of entities in an entity ent's neighbors belonging to class c_i, $i = 1, ..., L$, respectively, then the contextual type distribution of ent is a discrete distribution $d_{ent}(i) = (\frac{n_i}{n})$, where $n = \sum_i n_i$, $i = 1, ..., L$.

It is possible to explore the class hierarchy and the information content of classes to get a more comprehensive description of a context, but we find from an empirical study that the performance of this strategy is worse than the current simple strategy. A possible reason is that the contexts of the non-relevant entities become more similar to those of the relevant entities when incoporating the general superclasses in the class hierarchy, which makes the context information less discriminative.

The similarity of two contexts is simply defined as the intersection of the corresponding type distributions:

Definition 5 (Contextual Similarity) Given two entities ent_1 and ent_2 and their contextual type distributions d_{ent_1} and d_{ent_2}, their contextual similarity is defined as:

$$sim_c(ent_1, ent_2) = \sum_{i=1}^{L} \min(d_{ent_1}(i), d_{ent_2}(i)). \tag{11}$$

Similar to type inference, we define a relevance score for each candidate entity based on its contextual similarity to the positive entities. More formally:

Definition 6 (Contextual Relevance Score) Given a query node v, a candidate entity ent, and a set of positive entities $\{ent_1, ..., ent_n\}$, the contextual relevance score of ent is defined as:

$$s_c(ent|v) = \frac{\sum_{i=1}^{n} sim_c(ent, ent_i)}{n}. \quad (12)$$

The two types of implicit query information from user feedback: entity type and entity context can be computed efficiently online. They can be treated as additional features and plugged into the existing ranking function, e.g., Eq (1). We denote the weights of s_t and s_n as w_t and w_n, which can be either manually set or learned using a few training instances.

7. SEARCHING FROM SCRATCH?

After putting all the possible modifications together, we end up with a new ranking function which is better aligned with a user's search intent. The next step is to apply the new ranking function to the answers the user has not seen so far. For traditional relevance feedback in information retrieval, a second search is usually conducted from scratch or on the basis of the initial search [36]. However, it could be costly for graph querying as the potential match space is huge. Therefore, we explore an alterative option, that is, simply to re-rank the answer list from the initial search using the new ranking function. The problem is stated formally as follows: We have retrieved an initial list of top-k answers using the original ranking function F and generated a new ranking function $\tilde{F}$. Now there is a binary decision problem: (Plan A) Simply re-rank the k answers in the initial list (re-ranking) or (Plan B) Search from scratch with the new ranking function (re-searching). The former strategy is very efficient, but may lose some good answers. On the other hand, Plan B incurs a non-negligible time overhead, but has the potential to discover good answers missing from the initial top-k list. Obviously, this decision depends on many factors, and itself is a trade-off between answer quality and query response time.

In this section, we formulate this decision as a binary classification problem: Given a query, predict which query execution plan we shall choose. We propose an automatic method to build a ground-truth training set and learn a binary classifier on this training set. The key step is to build the training set, which takes two steps: (1) Feature Extraction, to convert a query into a feature vector, and (2) Label Assignment, to decide which class label (re-ranking or re-searching) we should assign to each query. After constructing a training set like this, the training of a binary classifier becomes straightforward.

Given a ground-truth query set, i.e., all relevant matches for each query are known, the construction process of a training set is as follows: For each query in the query set, (1) Send the query to the base graph query engine and fetch the initial top-100 answers using the original ranking function F; (2) Mark the top-10 answers as relevant or non-relevant according to the ground truth; (3) Run our GRF framework to learn a new ranking function $\tilde{F}$; (4) Re-rank the rest answers in the initial list using $\tilde{F}$ and let $\mathcal{L}_r$ be the new list (feedback removed); (5) Conduct a fresh search using $\tilde{F}$ and let $\mathcal{L}_s$ be this top-100 list (feedback removed); (6) Extract a feature vector for the query using the feature extraction strategy in Section 7.1; (7) Assign a class label (re-ranking or re-searching) to the query using the label assignment strategy in Section 7.2.

7.1 Feature Extraction

The goal of feature extraction is to characterize each query using a set of features which can help us decide which execution plan we should take for this query. Two underlying factors affect the decision making of this trade-off, *query ambiguity* and *query complexity*. If a query is ambiguous, the quality of the initial answer list is more likely to be poor, and the potential gain in effectiveness of re-searching is thus large. If a query is complex, it may take a long time to search, and the potential gain in efficiency of re-ranking is large. To capture these two factors, we identify three classes of features: query features, match features, and feedback features. Due to space limitations, we only give a high-level description of each feature class with intuitive examples.

Query Features Features in this class include query graph size, selectivity of terms, and the average number of terms on each query node. The intuition is to differentiate queries that are more specific and less ambiguous.

Match Features The initial search process and results can also give hints for the decision. Features in this class include the number of candidate nodes and the average match score of the initial matches. For example, if the nodes of a query have a lot of candidate nodes in the knowledge graph, then re-searching may incur a high time overhead. If the average match score of a query is low, it might imply that the query is ambiguous and the initial search result is poor, so we may need to search again with the new ranking function in the hope of discovering more relevant matches.

Feedback Features The feedback matches could also provide useful information. For example, the number of positive matches, the ranking of the positive matches in the initial answer list, and the average match score of the positive matches and the negative matches.

Putting all the features together, we are able to convert each graph query into a 18-dimensional feature vector. The next step is to assign a class label to each training query.

7.2 Label Assignment

The task of label assignment is to assign a class label to each query in the training set. The question is, if we know the search result and the time cost of both the query execution plans, which plan, re-ranking or re-searching, is more beneficial? Intuitively, if the quality gain of re-searching is large and it takes a reasonable time, re-searching is more favored. Here we propose a quantitative measure to take both quality and runtime into consideration, where a threshold can be employed to control the trade-off.

Suppose we have an effectiveness measure h to evaluate how good a result list is, e.g., Average Precision (AP), then the gain of doing re-searching for the query is defined as:

$$gain = \frac{h(\mathcal{L}_s) - h(\mathcal{L}_r)}{\log(1 + t_s - t_r)}, \quad (13)$$

where t_s and t_r are the time cost of re-searching and re-ranking, respectively. We then assign the class label of Q, $l(Q)$, using the following strategy:

$$l(Q) = \begin{cases} \text{re-searching}, & \text{if } gain > \tau \\ \text{re-ranking}, & \text{otherwise} \end{cases} \quad (14)$$

where $\tau \in \mathbb{R}$ is a pre-defined threshold controlling the trade-off.

After constructing a training set as above, we are ready to train a binary classifier to make decision for future queries. In Section 8 we use random forest as the classifier and show that our strategy can achieve a good trade-off between answer quality and query time. In practice, one can try different classifiers to select the best one.

8. EXPERIMENTAL EVALUATION

In this section, we empirically evaluate our methods. SLQ is used as the base graph query engine. We first describe the experiment design (Section 8.1) and then answer the following questions: (1) Can our GRF framework improve the search effectiveness of SLQ? If so, how much does each component contribute? (Section 8.2) (2) What is the impact of the hyper-parameters in GRF? (Section 8.3) (3) Can our learning-based algorithm lead to good trade-off between answer quality and query run time? (Section 8.4)

8.1 Experiment Design

Knowledge Graph DBpedia [18] is a popular knowledge graph extracted from Wikipedia. After indexing DBpedia 3.9[1] using SLQ, we end up with a knowledge graph containing $4.6M$ nodes and $100M$ edges. The ontology of DBpedia contains 529 classes which form a 8-layer class hierarchy.

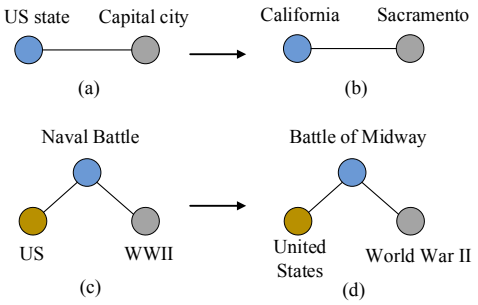


Figure 3: Exemplar queries and ground truth.

Graph Queries Plenty of benchmarks (e.g., TREC [4]) are available for researchers to evaluate their relevance feedback methods in document retrieval. Unfortunately, there is no widely-accepted benchmark for graph querying. Previous studies have resorted to ad-hoc evaluation. Here, we propose two methods to generate graph queries with ground-truth.

Our first query generating method capitalizes the list pages in Wikipedia. A list page contains structured information about a topic. For example, the page "List of States and Territories of the United States" contains a table about US states and their capital city, popularity, etc. We can therefore construct graph queries from such structured information. Figure 3(a) shows an example. Its answers can be extracted from the list page. We manually crafted 50 queries and denote this query set as WIKI. Most of them have two nodes and one edge.

Due to the miscellaneous ways of structured information representation in Wikipedia, it is hard to automatically construct graph queries. Therefore, we propose a second query generating approach which is more automated and thus more scalable. This approach leverages the YAGO ontology [31], a rather fine-grained ontology with $350K$ classes forming a 20-layer class hierarchy. It contains a lot of highly specific classes which can be converted to graph queries. For example, there is a YAGO class about "Naval Battles of World War II Involving the United States", from which we can formulate the graph query in Figure 3(c). Another advantage is that DBpedia entities are annotated with YAGO classes, which makes it feasible to build the ground truth[2]. Figure 3(d) shows a sample answer. We select 100 YAGO classes and convert them into ground-truth queries. This query set is denoted as YAGO. YAGO queries are more diverse in terms of structure: The number of query nodes range from 1 to 4, while the number of query edges range from 0 to 3. All the queries have multiple answers so that users have a chance to provide some positive feedback.

Experiment Pipeline Besides the knowledge graph and the graph query sets, we also need a way to obtain user feedback. Following the convention in IR [19], we simulate *explicit feedback* using ground truth, that is, we use the ground truth of a query to determine the relevance of several top-ranked matches from an initial search and use them as user feedback. For a given graph query, our experiment runs as follows: (1) Use SLQ to retrieve the initial top-100 matches. (2) Use the ground truth of the query to identify the relevant and non-relevant matches in the top-N_{fb}. (3) Take the result as user feedback to run GRF and obtain a new ranking function. (4) Finally, run SLQ with the new ranking function to retrieve a new list of top-100 matches. The feedback size N_{fb} is a hyper-parameter (typically set to 10, if not otherwise stated). Although we use explicit feedback as the default setting, we also evaluate GRF using *pseudo feedback*, i.e., the top N_{fb} initial matches are blindly treated as relevant, to test GRF's performance when feedback information is noisy or even erroneous. We choose Mean Average Precision at different cutoffs (MAP@K, $K = 1, 5, 10, 20$, etc.) as the main evaluation metric. For explicit feedback, feedback matches are removed before calculating MAP for the sake of fair comparison. We use paired Student's t test with $p = 0.05$ for significance test.

8.2 Overall Performance

The question we want to answer in this experiment is, can our GRF framework improve the search quality of SLQ? If so, how much contribution each component makes? We first experiment with explicit feedback on both query sets: Do model selection (choose values for hyper-parameters) on one query set and then test on the other. The experiment results are shown in Figure 4. The horizontal axis (K) indicates different cut-offs and the vertical axis shows MAP@K. We

[1] `http://wiki.dbpedia.org/Downloads39?v=2zd`

[2] YAGO classes are excluded from the following experiments.

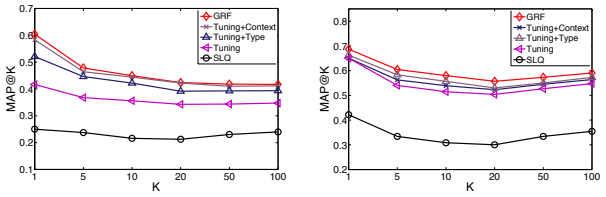


Figure 4: Performance of different GRF variants on: (a) WIKI, (b) YAGO.

Table 1: GRF vs. SLQ with varying query size.

Query Size	1	2	3	4
# Queries	7	62	25	6
SLQ	0.45	0.30	0.23	0.10
GRF	**0.56**	**0.57**	**0.56**	0.12

evaluate the contribution of the three components: query-specific tuning, type inference and context inference. GRF is able to bring a significant improvement in mean average precision. In comparison with the baseline SLQ, all GRF variants get significantly better performance. Moreover, the results demonstrate that all of the three components are useful and complement each other: query-specific tuning alone already results in a significant improvement, and adding the other two components further improves. The full GRF pipeline achieves the best performance on both query sets, improving over the baseline by 102% on WIKI and 86% on YAGO (MAP@20).

We also compare the performance of SLQ and GRF with varying query graph size (number of query nodes). We test on YAGO queries since they are more diverse in structure. Experiment results are shown in Table 1. MAP@20 is reported and statistically significant results are bold-faced. In comparison with SLQ, the performance of GRF is consistent across query sizes (1-3). An exception is when query size is 4. The quality of the initial search by SLQ is very poor and there are few relevant matches ranked top-10 in the initial answer lists. Due to the lack of positive feedback, the improvement of GRF is tiny. It is worth a further study for the situation where only negative feedback is available. This also includes the situation where a query only has one answer and the answer is not in the initial top list. Selective application of GRF based on query difficulty can also be investigated [5].

As a complementary experiment, we also evaluate GRF with pseudo feedback. The results are shown in Table 2. GRF still significantly outperforms SLQ when the feedback information is noisy or even erroneous.

8.3 Impact of Hyper-parameters

We evaluate four hyper-parameters: balance parameter λ, feedback size N_{fb}, and weights of the two relevance scores from type and context w_t and w_c. We only report the experiment results on WIKI. The observations on YAGO are similar and thus omitted for brevity.

Figure 5(a) shows the results of different λs in query-specific tuning. $\lambda = 1$ means we do not do re-weighting, which is the baseline SLQ. As we discussed, λ controls the balance between user feedback and the original ranking function. Experiment results show that a moderate value of λ is more appropriate. When λ is too small (e.g., 0.1), we overfit to the user feedback and cannot generalize well to unseen results. Therefore, regularization is helpful in this case to prevent overfitting. In the following experiments, λ will be set to 0.3.

Table 2: GRF vs. SLQ with pseudo feedback.

MAP@K	1	5	10	20	50	100
SLQ_WIKI	0.23	0.21	0.24	0.25	0.27	0.28
GRF_WIKI	**0.73**	**0.58**	**0.52**	**0.50**	**0.49**	**0.49**
SLQ_YAGO	0.40	0.35	0.33	0.32	0.36	0.39
GRF_YAGO	**0.82**	**0.66**	**0.60**	**0.57**	**0.58**	**0.61**

The feedback size N_{fb} also affects system performance. Intuitively, we should get better performance if we have more feedback information. In this experiment, feedback matches are not excluded before evaluation, because otherwise the experiment results of different N_{fb} will become not directly comparable. Figure 5(b) shows that, as N_{fb} increases, MAP increases in the mean time, which is as expected. But the gain of large N_{fb} (e.g., 20) is relatively small. Since a larger feedback size incurs a heavier burden for acquiring user feedback, $N_{fb} = 10$ seems a good setting.

Finally, we evaluate the impact of w_t and w_c. Basically, w_t and w_c specify how much weight we put on each relevance score. Figure 5(c) and Figure 5(d) show that a moderate value is more appropriate. When the weight of either component is too large (e.g., 10), we are overfitting to that component and system performance is therefore affected.

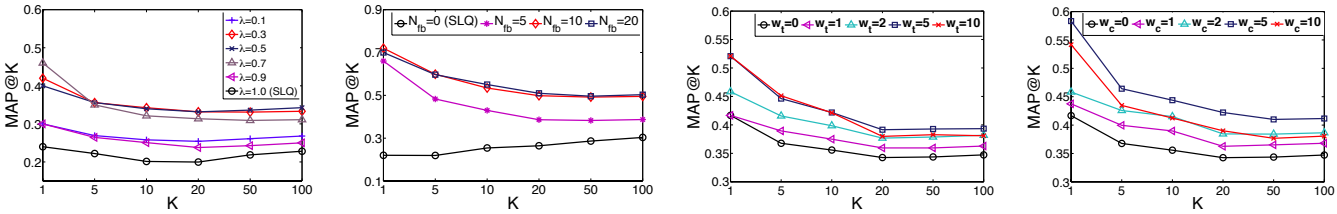
8.4 Answer Quality vs. Runtime

In this experiment, we examine the trade-off between answer quality and runtime (Section 7). Average Precision@20 is used as the effectiveness metric h. We experiment on YAGO and train a random forest using training sets generated with different τ. We employ the leave-one-out strategy for evaluation: For each query, we train a random forest on all the other queries and test on that query. The reported results are averaged over all the queries. Experiment results are shown in Figure 6. In this dataset, the re-ranking strategy achieves good performance: The answer quality does not decrease very much while the runtime is almost negligible. As shown by the results, the threshold τ can help make a good trade-off between answer quality and runtime.

9. RELATED WORK

Knowledge Graph Search. Various techniques have been proposed to search knowledge graphs. One popular search paradigm is structured search [3, 16], where queries are formulated using exact schema items (entity names, classes, relations) of the knowledge graph. But the usability of structured search is reduced due to the high and ever-growing heterogeneity of knowledge graphs. For this reason, keyword search [12, 23, 39] has been explored in order to improve accessibility. [35] gives a recent survey of the study in this line. However, users also lose the chance to specify query constraints to better express their information need.

Graph querying techniques [37, 17, 24] emerge as an alternative search technique. They allow users to (1) formulate queries using their own vocabulary, and (2) express query



(a) (b) (c) (d)

Figure 5: Impact of the hyper-parameters on WIKI: (a) balance parameter λ, (b) number of feedback matches N_{fb}, (c) weight of the type relevance score w_t, (d) weight of the context relevance score w_c.

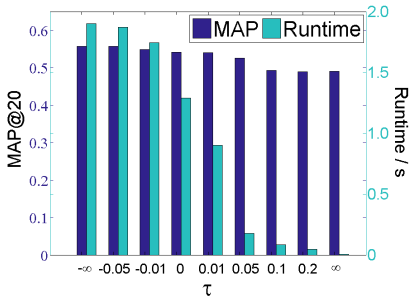


Figure 6: Answer quality vs. Runtime on YAGO

constraints via graph structure. Various ranking functions have been proposed. For example, [24] defines a ranking function which combines both semantic coherence and syntactic similarity, while NeMa [17] takes both syntactic similarity and the structural proximity of the matched nodes into account. However, all these ranking functions are generic. We propose to convert these generic ranking functions into query-specific ranking functions using user feedback.

The idea of searching for more entities based on a set of positive entities bears some similarity to entity search [21], where the goal is to find more entities given a set of exemplar entities. Another work similar in spirit to ours is [30], which employs pseudo-relevance feedback to improve keyword search over RDF graphs. To the best of our knowledge, our work is the first attempt to investigate relevance feedback in graph querying, which is quite different from keyword search.

Relevance Feedback in IR. Relevance feedback has been studied extensively in information retrieval. A relevance feedback method is usually designed for a specific retrieval model. For example, [38] incorporates relevance information into the language model, [28] works for the vector space retrieval model, and [27] employs relevance information to re-estimate the parameters in a probabilistic retrieval model. Therefore, they are not directly applicable to the graph query paradigm.

The idea of incorporating user feedback with the original model is not new. Traditional relevance feedback methods also combine the feedback information with the original query/model to produce the final ranking function (e.g., [28, 27, 38]). We employ regularization techniques to prevent overfitting, which is conceptually similar to [32], where a regularized EM algorithm is proposed for the language model. But our ways of regularization are quite different because of the different retrieval models.

New relevance feedback methodologies recently under exploration in information retrieval can potentially be applied to graph relevance feedback as well. [25] and [34] exploit the relations between retrieved results and define a query-specific ranking function on all the results, instead of on each individual result. [19] works to predict the optimal balance parameter between the original model and the feedback information for each query, instead of a fixed balance parameter for all the queries. Finally, [20] explores the idea of a non-uniform context by assigning higher weights to words closer to a matched keyword. The idea can also be applied to our GRF framework. A possible way is to treat the entities in the context of an entity differently based on their similarity or relation strength to the entity.

The efficiency issue of relevance feedback has also attracted some attention in IR [36, 10]. Their focus is to reduce the runtime of the second search. We approach this problem from a different perspective. Instead of always conducting a second search, we also consider the option of simply re-ranking the initial list. Our experiment results show that it can be a reasonable strategy: Depending on queries, the answer quality might not decrease too much.

10. CONCLUSIONS

In this work, we identified the limits of the generic ranking mechanism employed by existing graph querying techniques, and proposed to combine relevance feedback with graph querying in order to achieve query-specific ranking, i.e., GRF. We developed a novel GRF framework which works to tune the original ranking function as well as inferring additional information to enrich the query itself. We further identified an accompanying efficiency issue of GRF and proposed a classification mechanism to control the trade-off between answer quality and runtime. As verified by the experiments, our GRF framework can significantly improve the precision of a state-of-the-art graph querying technique, and make a good trade-off between answer quality and runtime in the mean time.

As the first attempt to study relevance feedback in graph querying, our work opens up an array of interesting future directions, e.g., discriminative feature mining from positive and negative feedback, and personalizing graph querying.

11. ACKNOWLEDGMENTS

This research was sponsored in part by the Army Research Laboratory under cooperative agreements W911NF-09-2-0053 and NSF IIS 0954125. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the

12. REFERENCES

[1] Google Knowledge Graph. http://www.google.com/insidesearch/features/search/knowledge.html.

[2] JOptimizer. http://www.joptimizer.com.

[3] SPARQL 1.1. http://www.w3.org/TR/sparql11-overview.

[4] TREC: relevance feedback track. http://trec.nist.gov/data/relevance.feedback.html.

[5] G. Amati, C. Carpineto, and G. Romano. Query difficulty, robustness, and selective application of query expansion. In *Advances in information retrieval*, pages 127–137. 2004.

[6] J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, pages 1533–1544, 2013.

[7] J. Berant and P. Liang. Semantic parsing via paraphrasing. In *ACL*, volume 7, page 92, 2014.

[8] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *SIGMOD*, pages 1247–1250, 2008.

[9] C. Carpineto and G. Romano. A survey of automatic query expansion in information retrieval. *ACM Computing Surveys*, 44(1):1, 2012.

[10] MA Cartright, J. Allan, V. Lavrenko, and A. McGregor. Fast query expansion using approximations of relevance models. In *CIKM*, pages 1573–1576, 2010.

[11] W. Eberle and L. Holder. Applying graph-based approaches to insider threat detection. In *Proc. of the 5th Annual Workshop on Cyber Security and Information Intelligence Research*, pages 206–208, 2009.

[12] S. Elbassuoni and R. Blanco. Keyword search over RDF graphs. In *CIKM*, pages 237–242, 2011.

[13] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. The vocabulary problem in human-system communication. *Comm. of the ACM*, 30(11):964–971, 1987.

[14] M. Gan, X. Dou, and R. Jiang. From ontology to semantic similarity: calculation of ontology-based semantic similarity. *The Scientific World Journal*, 2013.

[15] N. Jayaram, M. Gupta, A. Khan, C. Li, X. Yan, and R. Elmasri. GQBE: Querying knowledge graphs by example entity tuples. In *ICDE*, 2014.

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

[17] A. Khan, Y. Wu, C. C. Aggarwal, and X. Yan. NeMa: Fast graph search with label similarity. *VLDB*, 6(3):181–192, 2013.

[18] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. Kleef, S. Auer, and C. Bizer. DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 2014.

[19] Y. Lv and C. Zhai. Adaptive relevance feedback in information retrieval. In *CIKM*, pages 255–264, 2009.

[20] Y. Lv and C. Zhai. Positional relevance model for pseudo-relevance feedback. In *SIGIR*, 2010.

[21] S. Metzger, R. Schenkel, and M. Sydow. Aspect-based similar entity search in semantic knowledge graphs with diversity-awareness and relaxation. In *Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, volume 1, pages 60–69, 2014.

[22] D. Mottin, M. Lissandrini, Y. Velegrakis, and T. Palpanas. Exemplar queries: Give me an example of what you need. *VLDB*, 7(5), 2014.

[23] J. Pound, A. K. Hudek, I. F. Ilyas, and G. Weddell. Interpreting keyword queries over web knowledge bases. In *CIKM*, pages 305–314, 2012.

[24] J. Pound, I. F. Ilyas, and G. Weddell. Expressive and flexible access to web-extracted data: A keyword-based structured query language. In *SIGMOD*, pages 423–434, 2010.

[25] T. Qin, T. Liu, X. Zhang, D. Wang, and H. Li. Global ranking using continuous conditional random fields. In *NIPS*, pages 1281–1288, 2009.

[26] P. Resnik. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *JAIR*, 11:95–130, 1999.

[27] S. E. Robertson and K. S. Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27(3):129–146, 1976.

[28] J. J. Rocchio. Relevance feedback in information retrieval. *In The SMART Retrieval System: Experiments in Automatic Document Processing*, pages 313–323, 1971.

[29] I. Ruthven and M. Lalmas. A survey on the use of relevance feedback for information access systems. *The Knowledge Engineering Review*, 18(02):95–145, 2003.

[30] S. Shekarpour, K. Hoffner, J. Lehmann, and S. Auer. Keyword query expansion on linked data using linguistic and semantic features. In *Semantic Computing, 2013 IEEE Seventh International Conference on*, pages 191–197, 2013.

[31] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A core of semantic knowledge. In *WWW*, 2007.

[32] T. Tao and C. Zhai. Regularized estimation of mixture models for robust pseudo-relevance feedback. In *SIGIR*, pages 162–169, 2006.

[33] L. Terveen and D. W. McDonald. Social matching: A framework and research agenda. *ACM Trans. Comput.-Hum. Interact.*, 12(3), 2005.

[34] C. Wang, E. Yilmaz, and M. Szummer. Relevance feedback exploiting query-specific document manifolds. In *CIKM*, pages 1957–1960, 2011.

[35] H. Wang and C. C. Aggarwal. A survey of algorithms for keyword search on graph data. In *Managing and Mining Graph Data*, pages 249–273. 2010.

[36] H. Wu and H. Fang. An incremental approach to efficient pseudo-relevance feedback. In *SIGIR*, pages 553–562, 2013.

[37] S. Yang, Y. Wu, H. Sun, and X. Yan. Schemaless and structureless graph querying. *VLDB*, 7(7), 2014.

[38] C. Zhai and J. Lafferty. Model-based feedback in the language modeling approach to information retrieval. In *CIKM*, pages 403–410, 2001.

[39] Q. Zhou, C. Wang, M. Xiong, H. Wang, and Y. Yu. SPARK: adapting keyword query to semantic search. In *ISWC/ASWC*, 2007.

[40] X. S. Zhou and T. S. Huang. Relevance feedback in image retrieval: A comprehensive review. *Multimedia systems*, 8(6):536–544, 2003.

[41] B. Zong, R. Raghavendra, M. Srivatsa, X. Yan, A. K. Singh, and KW Lee. Cloud service placement via subgraph matching. In *ICDE*, 2014.

[42] L. Zou, R. Huang, H. Wang, J. X. Yu, W. He, and D. Zhao. Natural language question answering over rdf: a graph data driven approach. In *SIGMOD*, pages 313–324, 2014.