Unleashing the Power of Information Graphs

Matteo Lissandrini
University of Trento
ml@disi.unitn.eu

Davide Mottin
University of Trento
mottin@disi.unitn.eu

Themis Palpanas
Paris Descartes University
themis@mi.parisdescartes.fr

Dimitra Papadimitriou
University of Trento
papadimitriou@disi.unitn.eu

Yannis Velegrakis
University of Trento
velgias@disi.unitn.eu

ABSTRACT

Information graphs are generic graphs that model different types of information through nodes and edges. Knowledge graphs are the most common type of information graphs in which nodes represent entities and edges represent relationships among them. In this paper, we argue that exploitation of information graphs can lead into novel query answering capabilities that go beyond the existing capabilities of keyword search, and focus on one of them, namely, *exemplar queries*. Exemplar queries is a recently introduced paradigm that treats a user query as an example from the desired result set. In this paper, we describe the foundations of exemplar queries and the significant role of information graphs, and we present several applications and relevant research directions.

1. INTRODUCTION

The typical way of searching documents, objects or structures is through queries [1, 4]. Structured queries describe very accurately the objects of interest but they are hard to formulate. For this reason, several different search models have been studied in the past, including keyword search, similarity search, related search, personalized results, query refinement and query relaxation. However, these types of queries are vague and auxiliary information, such as knowledge bases, query logs or user profiles, is needed in order to improve the quality of the retrieved results [25]. One type of such auxiliary information is information graphs. Information graphs are graph structures like social graphs, knowledge bases, gene networks, etc. A common form of information graph is a *knowledge graph* in which nodes are entities, such as *Google* and *YouTube*, and edges are relationships between these entities, e.g., *Google acquired YouTube*.

A recently introduced query paradigm is *exemplar queries* [19]. Exemplar queries are particularly useful to cases in which the user knows one single element among those that are expected to be in the desired result set, and the system needs to infer the rest of the elements from it. For instance, a traditional keyword query on the World War II will traditionally return documents related to this war. Evaluating the query as an exemplar query will return many other big wars in history, such as the Vietnam War or the American Civil War. In other words, the user "query" is just an example of the elements and inter-relationships of interest that are expected to be returned by the search engine. Exemplar queries are particularly suitable for the case of an investigator, a lawyer, a reporter, a student, or a citizen that needs to perform a study on a topic to which she may not be familiar with, yet has as a starting point an element from the desired result set. Initial studies show that more than 80% of the users would benefit from a service implementing this paradigm [19]. As we discuss below, the exemplar query paradigm is flexible enough to be applied to many different data sources as long as there exists an effective method to convert them into information graphs.

Exemplar query answering is a multi-step task that evaluates an exemplar query and returns the set of results that are considered similar to those the user provided. The first challenge is that a user may not be able, or willing to provide all the details of the example, thus we need first to identify a subgraph of the information graph that better matches the user input. To do so one can exploit information graphs by identifying subgraphs that satisfy the user provided conditions. These subgraphs are referred to as *user samples*. Once the user samples have been retrieved, similar structures need to be found by employing graph similarity techniques. In order to provide different aspects of the results found in the previous step, the structures are refined by adding nodes and edges that belong to the information graph and are connected to those results. Finally, the results are ranked to present first those that more likely match the user's interests. If

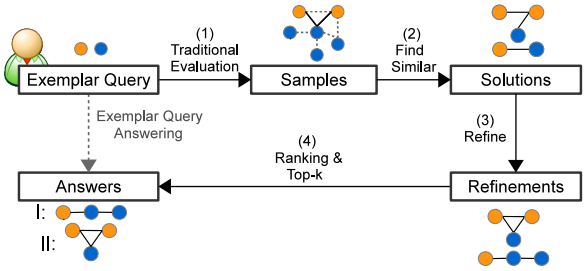

Figure 1: Exemplar query answering.

the number of these results is large then only the top-k can be produced and shown. These steps are graphically depicted in Figure 1 and have been implemented in the *XQ* system [20].

In the following, we provide a detailed description of how information graphs are used in these steps, and present a number of other applications scenarios in which informations graphs can be used.

2. EXEMPLAR QUERY ANSWERING

We define an *information graph* as a graph with labels on the edges and names as identifiers on the nodes. More specifically, an information graph is a pair $G : \langle N, E \rangle$, where N is the set of nodes and E is the set of edges. Each edge has a label l from the set $\mathcal{L}$ of infinite label names.

In *exemplar queries*, the information provided by the user is considered as a member of the desired answer set. Evaluating an exemplar query means discovering what are the main characteristics of that example and finding other similar structures in the information graph. Given a function $eval(Q_e)$ that evaluates the query Q_e on the information graph G and returns a set of samples, we define the evaluation of an exemplar query as follows.

Definition 1. *The evaluation of an* exemplar query Q_e *on an information graph* $G : \langle N, E \rangle$*, is the set* $\{a \mid \exists s \in eval(Q_e) \wedge a \approx s\}$*, where* a *and* s *are structures in* G *and the symbol* $\approx$ *indicates a similarity function.*

2.1 Finding Samples

A natural way of searching is keyword search. However, keyword queries are ambiguous in nature and as a result, many methods have been proposed to evaluate keyword queries into subgraphs of an information graph. Examples of such methods are semantic expansions [6, 23] and keyword to graph query translation [13, 25].

Evaluating a keyword query into a graph query requires several steps. The first is to split the query into chunks of one or more words. External information, such as document corpora, are usually exploited to mine the frequency of co-occurring words. The chunks are then assigned to candidate nodes in the graph by exploiting simple matching techniques on the node labels alongside some thesaurus. The next step connects the candidate nodes to form subgraphs. Since several subgraphs are found, probabilistic ranking (e.g., using random walks) is performed to retrieve the most likely *user samples*.

Relationships between concepts are often explicitly expressed in the user query. Some existing approaches explicitly elaborate on this fact either through similarities [2], or by employing mathematical models [3]. We explicitly take them into account to improve the quality of the sample retrieved from the information graph [14].

2.2 Finding Similar Structures

Traditional query answering on graphs [15, 27] is not directly applicable to exemplar queries, as it focuses on finding the best subset of nodes matching a given graph-query. Instead, in exemplar queries, structures *similar* to the user sample need to be retrieved in the information graph. We refer to these similar structures as *solutions*. Although any similarity measure can be used, a natural one is based on graph node-and-edge isomorphism [19]. While previous works are node-label preserving, i.e., they match both nodes and labels in the sample with those in the graph, exemplar query similarity preserves only the edge-labels and their structure.

Figure 2 illustrates a knowledge base and the sample: "*Google founded in Menlo Park acquired YouTube*" denoted as S in the figure. Exploiting the graph isomorphism similarity with the sample S, one can retrieve as solutions both $A1$:"*Yahoo! founded in S. Clara acquired del.icio.us*", and $A2$: "*GM founded in Flint acquired Opel*".

In the following, we briefly describe three algorithms for the identification of such structures. The first is an exhaustive algorithm that seeks in the graph the structures isomorphic to the user sample. The second is a fast pruning technique to restrict in advance the search space. Finally, the third algorithm is an approximate solution based on the Personalized PageRank measure. These techniques enable real-time exemplar query answering (i.e., responses in less than 1s), even when using the entire Freebase knowledge graph [19]).

Exhaustive solution. We assume as input a connected graph $S : \langle N_S, E_S \rangle$ that represents a sample. The notion of isomorphism is exploited to implement the similarity function $\approx$ in Definition 1, i.e., to find every subgraph $G' : \langle N_{G'}, E_{G'} \rangle$ of G that is

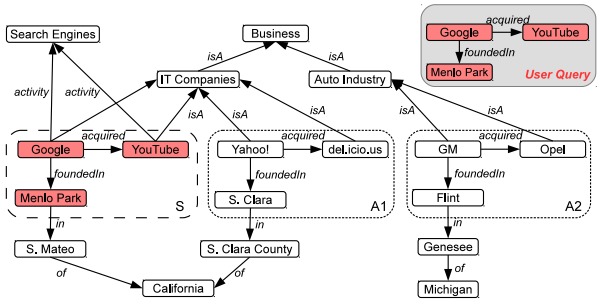

Figure 2: Answers to an exemplar query on a knowledge base

isomorphic to S. The set of all such subgraphs are the solutions. To do this, the sample S is compared with every other subgraph in the database. If an isomorphic graph is found, then it is added to the set of *solutions*. The algorithm initially starts from a node of the sample and finds a matching node in the information graph. It iteratively visits the nodes in the sample and in the graph until either all the nodes have been visited and a subgraph matching the sample has been found, or it is not possible to retrieve an isomorphic mapping. This algorithm is exponential in the number of query nodes.

Fast Exact Solution. Since the exhaustive solution is not efficient for large graphs, a better method for exploring the solution space is needed. The proposed method employs an efficient technique for comparing nodes, and an algorithm for effectively rejecting node pairs that do not participate in any isomorphic mapping. The idea is to store a compact representation of the neighborhood of each node. This representation is called d-neighborhood and referred to as $\mathbb{N}_d(n)$, i.e., nodes and edges that are at a fixed distance d from each node.

For every node in the database we compute a table consisting of the number of nodes that are reachable from that node at some specific distance and with a path ending with an edge labeled ℓ. In other words, for a node n, for every label ℓ and for every distance i we keep the cardinality of the set $W_{n,\ell,i} = \{n_1 | n_1 \xrightarrow{\ell} n_2 \vee n_1 \xleftarrow{\ell} n_2, n_2 \in \mathbb{N}_{i-1}(n)\}$.

This compact representation is then exploited in the algorithm through the concept of *simulation*. A graph simulates another graph if there exists a way to map each transition on the first graph with a transition in the second. Consequently, if a graph cannot simulate another graph, then they cannot be isomorphic. The main idea of our approach is to perform multiple simulations with the user sample on the database graph, comparing only the d-neighborhood representation of the nodes, without having to actually visit their neighborhood. The result is a method that operates much faster than the exhaustive solution, while providing the same, exact answers. While the algorithm is still exponential at worse, it is practically much faster than the exhaustive solution. The d-neighborhood can be computed and stored in advance in $\mathcal{O}(d|N||L|)$ space, where $L \subset \mathcal{L}$ is the finite set of edge-labels in G. Provided that the d-neighbor is sparse, efficient indexing techniques can be employed [15].

Approximate Solution. In order to further restrict the search space to a meaningful set of nodes that are likely to contain the top-k solutions for the user, a principled approach based on the Personalized PageRank (PPV) algorithm is considered. Since user preferences are expressed through the exemplar query, the PPV has to be computed over nodes in the information graph, instead of webpages. Instead of treating each edge equally, as in the original PageRank, and in order to better capture the semantics of the edge labels we propose the Adaptive Personalized PageRank Vector (APPV) method. APPV $\boldsymbol{v}$ is defined as the stationary distribution of the Markov chain with state transition given by $\boldsymbol{v} = (1-c)\boldsymbol{A}\boldsymbol{v} + c\boldsymbol{p}$, where $\boldsymbol{A}$ is a *non uniformly distributed* adjacency matrix and $\boldsymbol{p}$ is a preference vector computed from the user query. To compute this value fast, we apply an approach similar to the *weighted particle filtering procedure* [16], extended to take into account non-uniform edge weights. Finally, the algorithm returns the subset of G containing only those nodes with APPV score higher than a threshold. Isomorphic structures are then found in this subset.

2.3 Graph Query Refinement

If the user provides a general exemplar query with an exploratory intent, then query refinement is needed. Moreover, since a vague query usually retrieves a large number of *solutions*, query refinement can be used to group them offering a small set of more specific aspects that capture the structures contained in most of the solutions. These two common situations are the main motivations for the study of query refinements in information graphs. Query refinement has been studied in the web [7] and relational databases [21], but no straightforward adaptation exists with graph data sources.

A refined query is a more restrictive query than the original one, which retrieves a subset of the results of the original query. Therefore, finding refinements to a graph query means retrieving a set of supergraphs of the original query. Given a query Q we say that Q' is a reformulated query if (i) it is a connected graph, and (ii) there exists a subgraph

isomorphism from Q to Q'. Given a set of graphs $\mathcal{D} = \{G_1, ..., G_n\}$[1] and a query $Q : \langle N_Q, E_Q \rangle$ the results of the evaluation of Q over the set $\mathcal{D}$, is the set $\mathcal{R}_Q$ of subgraphs isomorphic to Q.

Given that the number of reformulations is exponentially large, only a meaningful subset should be returned. In order to retrieve relevant reformulations, the objective function sums two different contributions: the number of $\mathcal{R}_Q$ captured by the candidate reformulations (*coverage*), and the number of diverse results in each result set of the reformulated queries (*diversification*).

The coverage is a function *cov* that takes as input a set of reformulated queries R, and computes the number of results captured by R, that is, $cov(R) = |\bigcup_{Q \in R} \mathcal{R}_Q|$. The diversification is represented by a function *div* that takes into account the number of elements that are in the result set of Q_1 and not in Q_2, namely $div(Q_1, Q_2) = |\mathcal{R}_{Q_1} \cup \mathcal{R}_{Q_2}| - |\mathcal{R}_{Q_1} \cap \mathcal{R}_{Q_2}|$. The final score of a set of reformulated queries is the linear combination of coverage and diversity

$$f(R) = cov(R) + \lambda \sum_{Q_1, Q_2 \in R} div(Q_1, Q_2)$$

where $\lambda \in [0, 1]$ is a parameter that regulates the diversification of the result set R. The set R that maximizes the two factors of $f(R)$ is selected.

Fast Exact Solution. The optimization problem can be reduced to an instance of Max-Sum Diversification, known to be **NP**-hard. For this problem, there exists a $\frac{1}{2}$-approximation greedy algorithm [8] that runs linearly with respect to the input size. The greedy algorithm selects at each step the query that maximizes the *marginal potential gain*, i.e., the difference between the value of the function f, adding one extra element and the current value of f. Although the greedy algorithm works well in the normal set-covering formulation, in which the sets of elements are known in advance, the generation of the reformulations requires an exponential number of operations, as we consider all the possible edges that can be added to the current query Q.

Therefore, an algorithm that allows fast and effective pruning of the search space is necessary. The algorithm should remove the query refinements that definitely do not participate in the optimal solution. Since any reformulated query generates other reformulations just by adding one extra edge, the algorithm should find, for each reformulated query already computed, an upper bound to the marginal potential gain. This algorithm will then preserve the correctness of the approximation, yet allowing for an effective pruning of the unpromising reformulations. A further optimization can be introduced expanding at each step the reformulated query with the maximum upper bound.

[1]A set of graphs is obtained considering a neighborhood of the solutions found in the previous step.

2.4 Ranking Exemplar Results

In order to rank the exemplar query results, the structural similarity as well as the closeness of the solution from the sample are taken into account. For the former, a metric that is based on a vectorial representation of nodes using its neighborhood is used [15], and extended to capture the differences among nodes that emerge when taking into account the edge-labels of the neighbors, denoted with $\mathcal{S}$. For the latter, the APPV values are used, as they provide information about the distance of any node from the nodes of the sample, and consequently the distance of each solution from the sample. The final score is the linear combination

$$\rho(n_s, n) = \lambda \mathcal{S}(n_s, n) + (1 - \lambda)\boldsymbol{v}[n], \qquad (1)$$

where n_s is a node in the sample, n a node in the result, $\boldsymbol{v}[n]$ is the APPV score and λ is a user-defined parameter that regulates the amount of diversification in the results. For instance, in Figure 2 the ranking function will rank $A1$ higher, because it is closer to the sample S than $A2$.

3. APPLICATIONS

Exemplar queries find various applications in several different contexts. Below we discuss four such applications, namely, related queries, document search, seed-based search, and recommender systems.

3.1 Related Queries

Keyword queries are typically vague in their semantics, and a lot of work has been devoted in proposing alternative queries that generate results, which are more likely to match the user interests. *Query refinement* [18] and *query relaxation* [21] are two techniques that make a user query more specific, or more generic, respectively, in order to better match the user expectations. A similar concept is the concept of *related queries* [1]. The idea is to offer to the user a number of alternative queries that retrieve concepts that are related to those that the user asked about, or concepts that the user may be interested in finding more information about. Identifying related queries is a challenging task. Various methods, such as the exploitation of user query logs [1], large document corpuses [4] and knowledge bases [23] have been proposed. Exemplar queries can be seen as a complementary methodology that

suggests alternative queries. Having identified the user samples in a knowledge graph (e.g., Freebase) from the keyword query that the user provided, finding the related structures is like finding related concepts, from which queries can be formulated and then presented to the user as related queries [19].

3.2 Document Search

Document search is the core task of web search engines, as well as that of many document databases. Given a keyword query, the search engine retrieves the documents that are related to these keywords, which typically means that the documents have these keywords in their content. Current solutions are based on a plethora of different technologies, like topic-models [5] and click-models [10], and exploit various data sources, such as query-logs and knowledge bases. Exemplar queries can offer new ways to enrich this type of query results. Evaluating the keyword query through a traditional evaluation process, and at the same time through the exemplar query evaluation process described previously, will lead to a much richer answer set. This answer set will contain documents that may not include the original query keywords, but will involve concepts that are similar to those expressed in the original query. Entity linking [11, 22] can translate a document into a graph sample to be used in a knowledge graph as an exemplar query. This type of search has often been coined as semantic search, since the user is looking for semantically related documents, even if their direct structural relationship is not explicit. Furthermore, what is critical in this type of search is to be able to identify user intentions through the actions (e.g., previous searches) that a user has performed. This can be achieved by exploiting a graph representing previous knowledge on sequences of actions leading to the achievement of some goal [24].

3.3 Seed-based Search

Documents in document collections may be associated (related) in ways not always expressed through syntactic and semantic content similarity. Looking for this kind of similarity is encountered in many practical scenarios. For instance, in the case of technical blogs, in which users describe solutions to various problems that have faced, a user may be interested in finding a solution to a technical problem and uses a keyword search to identify responses from users that have faced different (but of similar nature) problems. Keyword based similarity will not work in most of such cases. Exemplar queries can help by considering a specific solution as an example and then searching of similar cases. To find such similarities, entity identification [11] can be used to turn the unstructured text document into a structured graph, and this structured query graph can then be matched against a knowledge graph.

Instead of seeing a document as a set of interconnected entities, one can see it as a set of segments, each one used to serve a specific communication message. A segment may be divided further into sub-segments that serve more specific purposes. At the end, the document, fragmented into segments and subsegments has been turned into a tree where the nodes represent the segments. Given a sample document, finding similar documents may turn to be a task of finding documents with the same segment structure or type of segment structure, even if the content of the segments are very different.

3.4 Recommender Systems

Recommender systems try to make item suggestions from an item set, or rank the results of a user query, based on the personal preferences of the user (according to her previous selections in the interaction with the system). This kind of decisions are based on some form of similarity to the properties of the items in the previous selections, as this is computed using a number of different metrics [26]. One can formulate the set of all the different properties of the items and organize them into some graph structure that represents their one-to-one relationships either based on their nature, or on the degree of preference by the various users. Then given some selection made by the user, additional suggestions of similar items can be made, through an exemplar query evaluation method that uses the item features graph as an information graph.

4. EXTENSIONS

The exemplar query answering framework presented previously can be extended in several directions, in order to offer more advanced services.

Multiple exemplar queries. The user may provide multiple examples of the same desired result set and this information can be used to more accurately identify the user needs. Although multiple query processing has been studied in relational databases [9], the exemplar queries context is completely different: in this earlier work, the results for a single, or multiple queries do not change, while in exemplar queries the results become more specific as the number of example queries increases. Moreover, the straightforward solution that computes the intersection between the result sets is not directly applicable when there is no common edge labels in the input samples. A recent study computes the rele-

vant neighborhood of the samples in order to find intersections among them [12]. However, this does not provide a clear semantics for multiple exemplar queries, thus the problem needs to be investigated further.

Approximate exemplar queries. Assume that a user looking for companies in California provides the query *"Google, S. Mateo"*. The identified sample connects *Google* to *S. Mateo* using the edges *foundedIn, in* as in Figure 2. The query, through graph isomorphism, retrieves companies founded in the bay area, that does not perfectly match the user intent. Approximate query answering could solve the problem, retrieving companies not only founded in California but also based in California, employing some similarity between edges. Therefore, extending the similarity measure to handle looser matchings is important. Several adaptations of the current task can be employed, such as approximate queries [17] and bi-simulation instead of graph isomorphism. Due to the large number of results, graph indexing techniques have to be adopted in order for the system to provide timely answers.

Personalization and dynamic ranking. The ranking function presented earlier does not take into account user preferences. User preferences can be mined from query logs, or other sources, and offer the basis for personalized results ranking. The Personalized PageRank (Section 2.2) can be extended to include any probability in edges and nodes, thus reflecting the preferences of the user. Moreover, without employing any approximation, the computation of the node similarity based on the neighborhood can take into account various preferences.

5. CONCLUSIONS

Information graphs are valuable and important source of information, as they embed knowledge in a simple graph structure. The use of information graphs to answer exemplar queries opens interesting research directions that cannot be addressed by existing technologies. This synergy between exemplar queries and information graphs forms the basis of the presented exemplar query answering framework. We argue that this framework is powerful and versatile, able to improve existing solutions by enriching the expressiveness and the ease of use of many search systems, and to increase user satisfaction. Finally, we discuss how this framework can be applied to several different scenarios, such as related queries, document and seed-based search that constitute the basis of modern search engines.

References

[1] A. Anagnostopoulos, L. Becchetti, C. Castillo, and A. Gionis. An optimization framework for query recommendation. In *WSDM*, 2010.

[2] S. Bergamaschi, E. Domnori, F. Guerra, R. Trillo Lado, and Y. Velegrakis. Keyword search over relational databases: a metadata approach. In *SIGMOD*, 2011.

[3] S. Bergamaschi, F. Guerra, S. Rota, and Y. Velegrakis. A hidden markov model approach to keyword-based search over relational databases. In *ER*, 2011.

[4] S. Bhatia, D. Majumdar, and P. Mitra. Query suggestions in the absence of query logs. In *SIGIR*, 2011.

[5] D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3, 2003.

[6] R. C. Bodner and F. Song. Knowledge-based approaches to query expansion in information retrieval. In *Canadian AI*, 1996.

[7] P. Boldi, F. Bonchi, C. Castillo, and S. Vigna. Query reformulation mining: models, patterns, and applications. *IR*, 14(3), 2011.

[8] A. Borodin, H. C. Lee, and Y. Ye. Max-sum diversification, monotone submodular functions and dynamic updates. In *PODS*, 2012.

[9] U. S. Chakravarthy and J. Minker. Multiple query processing in deductive databases using query graphs. In *VLDB*, 1986.

[10] F. Guo, C. Liu, A. Kannan, T. Minka, M. J. Taylor, Y. M. Wang, and C. Faloutsos. Click chain model in web search. In *WWW*, 2009.

[11] X. Han, L. Sun, and J. Zhao. Collective entity linking in web text: a graph-based method. SIGIR, 2011.

[12] N. Jayaram, A. Khan, C. Li, X. Yan, and R. Elmasri. Querying knowledge graphs by example entity tuples. *CoRR*, abs/1311.2100, 2013.

[13] G. Kasneci, F. M. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum. NAGA: Searching and Ranking Knowledge. In *ICDE*, 2008.

[14] Z. Kefato, M. Lissandrini, D. Mottin, and T. Palpanas. Keyword query to graph query. *DISI Technical Report*, 2013.

[15] A. Khan, N. Li, X. Yan, Z. Guan, S. Chakraborty, and S. Tao. Neighborhood based fast graph search in large networks. In *SIGMOD*, 2011.

[16] N. Lao and W. W. Cohen. Fast query execution for retrieval models based on path-constrained random walks. In *KDD*, 2010.

[17] W. Lin, X. Xiao, J. Cheng, and S. S. Bhowmick. Efficient algorithms for generalized subgraph query processing. In *CIKM*, 2012.

[18] C. Mishra and N. Koudas. Interactive query refinement. In *EDBT*, 2009.

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

[20] D. Mottin, M. Lissandrini, Y. Velegrakis, and T. Palpanas. Searching with xq: the exemplar query search engine. In *SIGMOD*, 2014.

[21] D. Mottin, A. Marascu, S. B. Roy, G. Das, T. Palpanas, and Y. Velegrakis. A probabilistic optimization framework for the empty-answer problem. *PVLDB*, 6(14), 2013.

[22] D. Mottin, T. Palpanas, and Y. Velegrakis. Entity ranking using click-log information. *IDA J.*, 17(5):837–856, 2013.

[23] V. M. Ngo and T. H. Cao. Ontology-based query expansion with latently related named entities for semantic text search. In *IJIIDS*, volume 283. 2010.

[24] D. Papadimitriou, G. Koutrika, Y. Velegrakis, and J. Mylopoulos. Goals in Social Media, Information Retrieval and Intelligent Agents. In *Proceedings of ICDE*, 2015.

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

[26] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, 2001.

[27] X. Yan, P. S. Yu, and J. Han. Graph indexing: a frequent structure-based approach. In *SIGMOD*, 2004.