Exemplar Queries: Give me an Example of What You Need

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

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

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

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

ABSTRACT

Search engines are continuously employing advanced techniques that aim to capture user intentions and provide results that go beyond the data that simply satisfy the query conditions. Examples include the personalized results, related searches, similarity search, popular and relaxed queries. In this work we introduce a novel query paradigm that considers a user query as an example of the data in which the user is interested. We call these queries *exemplar queries* and claim that they can play an important role in dealing with the information deluge. We provide a formal specification of the semantics of such queries and show that they are fundamentally different from notions like queries by example, approximate and related queries. We provide an implementation of these semantics for graph-based data and present an exact solution with a number of optimizations that improve performance without compromising the quality of the answers. We also provide an approximate solution that prunes the search space and achieves considerably better time-performance with minimal or no impact on effectiveness. We experimentally evaluate the effectiveness and efficiency of these solutions with synthetic and real datasets, and illustrate the usefulness of exemplar queries in practice.

1. INTRODUCTION

Traditional query answering is about finding the structures in a data repository that satisfy the query conditions [2, 7, 8, 12, 17, 32]. Recent advances in information and communication technologies have brought query answering systems to the general public, driving a significant effort for simplifying these systems to a level that allows them to be used by the average user. New techniques for answering simpler, less structured and less specific queries, have attracted considerable attention [6]. This is because the average users are typically not accustomed to the technicalities of the query language, neither its capabilities, which makes it hard to provide a full specification of the elements of interest. To cope with these situations, query answering systems have employed techniques such as query relaxation [22], semantic enhancements [5], statistics-driven query answering [14], log-based analysis [9, 24], and others.

A hidden assumption behind all the aforementioned techniques is that the user is aware of the characteristics of the structures of interest and can (at least partially) describe them in the query. We advocate here that there are many practical scenarios in which this is not the case. We are interested in those cases where the user knows one single element among those that are expected to be in the desired result set, and we would like to study ways to infer the rest of the elements from this. In other words, the user "query" works as an example of what the elements of interests that are expected to be returned by the search engine are. We call this novel query paradigm *exemplar queries* to emphasize its different nature from those previously mentioned and the new evaluation methods they require.

The notion of exemplar queries reminisces the well-known notion of query by example (QBE) [35], yet, it is fundamentally different. In QBE, the user query is also an example, but is used simply to communicate to the query evaluation engine the conditions in a more user-friendly way. In some sense, QBE works like a wildcard query. In contrast, the query in our case is rather a sample from the desired set, indicating the type of elements that are expected to be in the results. These elements may have characteristic properties different from those mentioned in the user query, simply because their similarity to the example that the user query provides may be based on characteristics that are not explicitly stated in the query. Our approach is also different from query relaxation [22, 23], which aims at producing more generic versions of a query. As in the case of QBE, these queries have some of the properties of the original query at their core.

Exemplar queries find important practical applications in information searching. They are particularly suitable for the case of a student, a curious citizen, an investigator, a lawyer or a reporter that needs to perform a study on a topic to which she may not be familiar, but has as a starting point an element from the desired result set. Exemplar queries can form the basis of a new form of search engines that uses them as the main query evaluation mechanism, or they can be used to enhance the services that existing search engines are currently offering. In particular, in parallel to the query evaluation a search engine performs, the query can also be seen as an exemplar query and be evaluated as such. These results can be appended to the results the search engine generates, increasing the probability to capture the user's

Proceedings of the VLDB Endowment, Vol. 7, No. 5

intent. Alternatively, the results of the exemplar query evaluation can be modeled as a set of queries, and then appended in the list of "related/additional queries" that most modern search engines are currently suggesting to their users. For instance, a query on the World War II will typically return documents related to this war. Evaluating the query as an exemplar query will result to many other big wars in history. Then, documents related to these wars can be added in the result set, or queries retrieving data about these wars can be added in the list of the related searches that the search engine suggests to the user.

For the evaluation of exemplar queries we consider two steps. The goal of the first step is to evaluate the user query and identify in the data repository the structure that the user is describing in the query. Once this structure has been identified, in the second step we examine the data store to find similar structures. Our approach is principled and does not depend on the data store model. In this study though, we focus on the case where the data store is based on a graph data model with labels on the edges. In order to find similar structures, we naturally use a version of graph similarity (in particular, graph-isomorphism), but in a way that takes into consideration also labels on the edges. Furthermore, we are interested only on the k most promising results. Traditional query answering on graphs [19, 20, 34], that focuses on finding the best subset of nodes matching a given graph-query, has no straightforward time-efficient adaptation to allow the retrieval of the top-k most similar subgraphs based on our form of similarity. The brute-force solution is exponential in nature, so we devised an efficient iterative pruning schema that pre-computes a representation of the neighborhood of each node, using only the information on the edges. We demonstrate that this algorithm is exact, i.e., it preserves the quality of the answers, while significantly reducing computation time. We also propose an approximate algorithm, which prunes the search space, keeping only the subgraph portion that is closer to the user-query, i.e., contains the top-k answers. We show that this heuristic works very well in practice, with no significant compromise on the quality of the results.

Our contributions can be summarized as follows: (i) we introduce and formally define a novel form of query answering, referred to as *exemplar queries*, that treats a query as a sample from the desired result set; (ii) we study exemplar queries for graph-based models, and devise a similarity metric that takes edge-labels into consideration; (iii) we propose two algorithms to compute the exact solution, a baseline and an optimized one, and we further describe an approximation algorithm with significant efficiency gains and minimal effect on quality; (iv) we experimentally show that existing approaches either fail to produce correct exemplar query evaluations, or they do so in a much longer time, that makes them inapplicable for online applications; finally, (v) we perform a thorough experimental evaluation, using the largest multigraph ever used (freebase) in this field, that demonstrates the efficiency of our solution, and a user-study that validates the usefulness of exemplar queries.

The remainder of the paper is structured as follows. Section 2 provides a motivating example while Section 3 defines formally the notion of exemplar queries, and an instantiation of the problem on a graph-based data model. Sections from 4 to 7 provide algorithmic solutions to this instantiation of the problem. Section 8 describes our experimental evaluation and findings. Finally, we present the related work in Section 9, and conclude in Section 10.

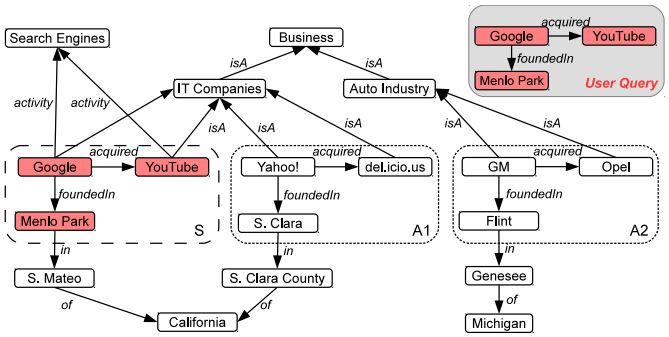

Figure 1: Exemplar Query Evaluation

2. MOTIVATING EXAMPLE

Consider a university student who has been given an assignment to perform a study on company acquisitions in the Bay area. The student is not really an expert in the field, and not familiar with the related terminology. Writing a query with the terms "acquisitions" and "Bay Area" will in the best case return documents talking about the topic of acquisitions, and also mentioning the Bay area. An article on the takeover of del.icio.us by Yahoo! may not be returned if the actual words of acquisition and Bay area are not explicitly mentioned in the text.

The student knows that a good case of the type of acquisition she is looking for is the one of YouTube by Google. Thus, she issues the query: *"Google founded-in Menlo Park acquired YouTube"*. The search engine typically responds with results related to Google, Menlo Park, and Youtube, but will not return anything related to an acquisition of del.icio.us by Yahoo!. If there is a significant number of users that have performed similar searches in the past, an analysis of the query logs may reveal that information and the search engine may be able to propose, in the related searches section, queries on Yahoo! and del.icio.us. (A simple test in existing search engines reveals that this is not actually happening.) Relaxing one or more of the query conditions does not help in a significant way, since the results are still focused around the Google case.

Consider now a second candidate answer for the user query: Opel that was acquired by General Motors (GM). Among the Yahoo!-del.icio.us and GM-Opel, it is more likely that the former is among the company acquisitions that the user is interested in, and not the latter. This is because even though Yahoo! was founded in a different city than Google, that city is still in California (just like with Google), while the city that GM was founded is in Michigan. Furthermore, the example of Google-YouTube that the user provided is about IT companies, and so are the Yahoo!-del.icio.us, while GM-Opel belong to the automotive industry.

Thus, there is a need to devise a method for inferring the set of elements that the user is interested in from a single sample (of that set), provided by the user.

3. PROBLEM STATEMENT

Achieving the required functionality seems to be a two-step process. The first is to identify in the data repository the structure that the user is referring to in the query, i.e., those that represent the sample that the user already

knows to be part of the desired result set. This can be easily achieved using traditional query evaluation techniques that identify in the data repository the structures satisfying the specifications set in the user query. We denote the results of this type of evaluation of a query Q as $eval(Q)$ and refer to it as the *user sample*.

The second step is to find the remaining structures of interest for the user based on the structure that has been identified in the first step. Note that there is a query that describes all these structures that the user is looking for, it is just that she is not aware of that query or is not in a position to describe it. Thus, it is natural to assume that all the structures of interest have some commonalities, especially to the one that the user provided as an indicative example. As such, we are interested in finding similar structures to the results of the first step, and return these results as an answer to the user-provided query.

We refer to this new query paradigm as *exemplar queries* and the results of their evaluation as *relevant answers*.

Definition 1. *The evaluation of an* exemplar query Q_e *on a database D, denoted as $xmpEval(Q_e)$, is the set $\{a \mid \exists s \in eval(Q_e) \wedge a \approx s\}$, where a and s are structures in D and the symbol $\approx$ indicates a similarity function.*

Looking for structures similar and not exact to those explicitly described in the query, reminisces query relaxation. Yet, it is different. In query relaxation, one or more of the query conditions are relaxed, so the results in the answer set are elements that satisfy some of the conditions of the user query. The desired results in our case may satisfy a small number of the query conditions, since the similarity to the structure specified by the user query may be based on characteristics that are different to those mentioned in the user query.

Note that the definition of exemplar queries is independent of the data model, of the query form, of the retrieved results and of the similarity function. As long as there is a standard query evaluation methodology and some similarity function that can be used that fits a specific use case, the exemplar queries can be answered. This allows maximum flexibility and the ability to use exemplar queries in a wide range of different applications. We are particularly interested in applying exemplar queries in cases where the data is highly heterogeneous and have some relaxed structure. For that reason we have chosen to use a flexible data model, a simple query form, a traditional query evaluation that is based on graph node & edge isomorphism, and a very generic similarity function that is based on edge label-preserving similarity on graphs.

For the representation of the data we consider a flexible entity-based data model [11] that can easily represent various forms of heterogeneous knowledge. In particular, we assume an infinite set of labels $\mathcal{L}$ and of values $\mathcal{V}$. The set $\mathcal{V}$ consists of an infinite set of atomic values $\mathcal{T}$ and of object identifiers $\mathcal{O}$, i.e., $\mathcal{V}=\mathcal{T}\cup\mathcal{O}$. An *object* is a representation of a real world entity or concept and is modeled through an object identifier and a set of attributes for that identifier that model characteristic properties of the real world entity or concept. An *attribute* of an object $o\in\mathcal{O}$ is a triple $\langle o, l, v\rangle$, where $l\in\mathcal{L}$ and $v\in\mathcal{V}$.

A database is a finite collection of objects, alongside a finite set of attributes for these objects. The attributes are either connecting the objects or specify some characteristic properties of them.

Definition 2. *A* database D *is a pair $\langle O, A\rangle$ where $O\subseteq\mathcal{O}$ and $A\subset O\times\mathcal{L}\times(O\cup\mathcal{T})$, both finite.*

A database can be represented as a graph where every object or atomic value in the database is represented as a node and every attribute as a labeled edge from the node representing the object of the attribute to the node representing its value. Thus, we can equivalently say that a database $\langle O, A\rangle$ is a graph $G(N, E)$, also denoted as $\langle N, E\rangle$, where the set of nodes N is the set $\{n \mid n\in O \vee \exists\langle n', l, n\rangle\in A\}$ and the set of edges E is the set $\{n \xrightarrow{l} n' \mid \langle n, l, n'\rangle\in A\}$. The expression $n \xrightarrow{l} n'$, denotes an edge from node n to node n' labeled l. We also say that two nodes n_1, n_2 are *equivalent*, and denote it as $n\equiv n'$, if they represent the same atomic value or the same object, i.e., the identifiers of the objects they respectively represent are the same.

Definition 3. *A database D is edge-preserving*[1] isomorphic *to a database D', denoted as $D\simeq D'$, if there is a bijective function μ from the nodes of D to the nodes of D' such that for every edge $n_1\xrightarrow{l}n_2$ in D, the edge $\mu(n_1)\xrightarrow{l}\mu(n_2)$ is in D'.*

A query is traditionally an expression describing a set of objects alongside a set of conditions they need to satisfy. These conditions describe certain characteristics of these objects and the relationships they may have among them. We make the natural assumption that the objects referenced in a query are somehow all connected, otherwise the query expression would actually constitute two independent queries. Since a query describes a set of objects with attributes, i.e., properties and relationships among them, it can also be seen as a database and consequently represented as a connected graph. Answering a query on a database means finding the database structures that satisfy the query specification. By the term database structures we mean a set of objects and a set of attributes for these objects. In graph terms, answering a query means finding the subgraphs in the database that have a structure like the one of the graph representation of the query. The set of these subgraphs constitutes the answer set of the query.

Definition 4. *A* query Q *is a database whose graph representation is a connected graph. An* answer *to a query $Q{:}\langle N_Q, E_Q\rangle$ on a database D is any subgraph $D'{:}\langle N_{D'}, E_{D'}\rangle$ of D that is isomorphic to Q, i.e., $D'\simeq Q$, and $\forall n_Q\in N_Q$, $n_{D'}\in N_{D'}$: $\mu(n_Q)=n_{D'} \Rightarrow n_Q\equiv n_{D'}$. The set of all such subgraphs, denoted as $eval(Q)$, is referred to as the* answer set *of the query.*

Note that our implementation of exemplar queries can also be used in cases where queries are flat keyword queries, provided that they are first translated to some structured form. This task is outside of the scope of this work, but there is already a large amount of literature [5] on that topic.

Regarding the similarity function, although multiple different forms of similarity can be used we consider edge label-only isomorphism as the implementation of this similarity, since we found it to be most natural for our purposes. Note that the similarity between structures may range from very

[1]In the rest of the document we will be dropping the part "edge-preserving"

Algorithm 1 XQ

Input: Database D: $\langle N, E\rangle$
Input: User Query Q
Output: Set of relevant answers $\mathfrak{Q}$
1: $\mathfrak{Q} \leftarrow \emptyset$
2: $S \leftarrow eval(Q)$
3: $n_s \leftarrow selectARandomNode(S)$
4: **for each** $n \in N$ **do**
5: $A \leftarrow$ FindIsomorphicSubgraph(S, n_s, D, n)
6: **if** $A \neq \emptyset$ **then**
7: $\mathfrak{Q} \leftarrow \mathfrak{Q} \cup A$
8: $Rank(\mathfrak{Q})$
9: **return** $\mathfrak{Q}$

high to very low with no clear understanding on where one should stop. For this we will be interested in finding only the k most similar structures, i.e., the top-k, and return them ranked.

Example 1. *Consider the example described in Section 2 and the portion of the database illustrated in Figure 1. The user query (the exemplar query) is the one shown at the top right corner of the figure. The evaluation of that query on the database results to the user sample that is indicated in the database with the dashed box labeled S. Searching for similar structures (edge-isomorphic structures) to this user sample, results to the two structures indicates with the dotted line boxes labeled $A1$ and $A2$, that serve as the relevant answers to the exemplar query. Among the two relevant answers, the neighborhood of $A1$ has more nodes and edges in common to the user sample S, for instance, the* `IT Company`*, the* `Search Engine` *and the* `California`*, than those that the neighborhood of $A2$ has in common, hence, $A1$ should be ranked higher than $A2$.*

Since the first step of the exemplar query evaluation is a standard search in a graph database for a graph (the user query) we will not spend more time on this. Instead, we focus on the implementation of the second step, which is to devise a method that from a given subgraph (the user sample) finds efficiently other edge-isomorphic subgraphs (the relevant answers) and ranks them based on their neighborhood. The challenging part of this is that there is no clear limit on how large neighborhood to consider, apart from the whole database itself. In our implementation we have considered Freebase, which is one of the largest knowledge graphs available nowadays. Existing works on graph similarity assume search on a large number of small graphs, but searching on a very large graphs in the form we consider here has not been considered extensively.

4. THE BASIC XQ ALGORITHM

Once the first step of the exemplar query evaluation has been performed and the user *sample* S has been identified in the database D, the set of similar to it structures will have to be discovered. This similarity is based on graph-isomorphism on the edge labels. To do so, the user sample S will have to be compared with every other subgraph in the database. Instead of considering the exponential number of subgraphs in the database, a node n_s from S is randomly selected to serve as a seed. Then all the nodes in the database D are considered, one at a time. For each such node n, it is checked if a subgraph that contains n and is isomorphic

Algorithm 2 IterativePruning

Input: A database $D : \langle N, E\rangle$
Input: A user sample $S : \langle N_S, E_S\rangle$
Output: A set of candidate mappings $\mu \subseteq N_S \times N$
1: $\mathbb{N}_d^S \leftarrow d$-neighborhood of S
2: Vis $\leftarrow \emptyset$ ▷ Visited nodes
3: $n_{min} \leftarrow \arg\min_{n \in N_S} Sel(n)$
4: $C \leftarrow \{n_{min}\}$ ▷ Query candidates
5: $\mu(n_{min}) \leftarrow \{n | \mathbb{N}_d^S(n_{min}) \subseteq \mathbb{N}_d(n)\}$
6: **for each** $n_s \in C$ **do**
7: **if** $n_s \xrightarrow{\ell} n'_s \in E_S$ and $n'_s \notin$ Vis **then**
8: $\mu(n_s) \leftarrow \mu(n_s) \setminus \{n | n \not\xrightarrow{\ell} n_1, n \in \mu(n_s)\}$
9: $\mu(n'_s) \leftarrow \{n_1 | n \xrightarrow{\ell} n_1, n \in \mu(n_s), \mathbb{N}_d^S(n'_s) \subseteq \mathbb{N}_d(n_1)\}$
10: **else if** $n'_s \xrightarrow{\ell} q \in E_S$ and $n'_s \notin$ Vis **then**
11: $\mu(n_s) \leftarrow \mu(n_s) \setminus \{n | n_1 \not\xrightarrow{\ell} n, n \in \mu(n_s)\}$
12: $\mu(n'_s) \leftarrow \{n_1 | n_1 \xrightarrow{\ell} n, n \in \mu(n_s), \mathbb{N}_d^S(n'_s) \subseteq \mathbb{N}_d(n_1)\}$
13: $C \leftarrow C \cup \{n'_s | n_s \xrightarrow{\ell} n'_s \vee n_s \xleftarrow{\ell} n'_s\}$
14: $C \leftarrow C \setminus \{n_s\}$
15: Vis $\leftarrow$ Vis $\cup \{n_s\}$

to S can be constructed. If such a graph is found, then it is added in the result set, i.e., the set of *relevant answers*. At the end of this procedure the relevant answers are sorted and returned all or only the top-k as an answer to the exemplar query. The sorting task is studied in details in Section 7. The pseudo-code of the above steps is described in Algorithm 1.

The construction of the isomorphic subgraphs (line 5 in Algorithm 1) is done by initially considering a graph G consisting only from the node n_s and a subgraph T consisting only from node n, and assuming that an isomorphic function maps n_s to n. Then iteratively trying to expand the subgraphs G and T with edges from S and D respectively such that the resulted subgraphs remain edge-isomorphic. If after a number of steps, the graph G becomes equal to S, then the graph T is one of the answers.

5. AN EFFICIENT EXACT SOLUTION

Searching for possible matches of the user sample in the entire database, as the Algorithm XQ requires is definitely an expensive operation. Sub-graph isomorphism is known to be an NP-complete problem. One can exclude the edges with labels that do not appear in the user sample, but the complexity and the number of operations that need to be done will still be high.

To improve the performance, we propose an effective way to prune the search space, i.e., the list of database nodes we have to match to the nodes of the user sample in order to find isomorphic structures, leading to a new algorithm FastXQ. For this we devise an efficient technique for comparing nodes, and an algorithm for effectively rejecting pairs of nodes that are bound to not participate in any isomorphic mapping, we call this algorithm IterativePruning. Although this technique may lead to false positives, the schema is effective and reduces significantly the search space. The false positives are subsequently removed by running the traditional isomorphic verification algorithm on them.

For the first, inspired by [19], we devise a technique that is meant to represent the neighborhood in a compact way,

and to match the nodes in advance without the need to examine all the nodes in the graph. In more details, the idea is to store in advance a compact representation of the neighborhood of each node, i.e., nodes and edges that are at a fixed distance d from each node. This provides an effective way to compare nodes, allowing the pruning to remove the non-matching nodes without having to actually visit their neighborhood.

A basic concept of our approach is also the notion of neighborhood. We call d-neighbor of a node n a node that is reachable from n in at most d steps, i.e., the shortest path from n to this node is no longer than d.

Definition 5 (d-neighbor)**.** *Let $n \in N$ be a node of a database $D = \langle N, E\rangle$. The node $n_i \in N$ is a d-neighbor of n if there exists a shortest path from n to n_i of length at most d. The d-neighborhood of n, denoted as $\mathbb{N}_d(n)$, is the set of d-neighbors of n.*

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 a label ℓ. In other words, for a node n, for every label ℓ and for every distance i we keep the cardinality of the set $W_{n,l,i}$, where

$$W_{n,\ell,i} = \{n_1 | n_1 \xrightarrow{\ell} n_2 \vee n_1 \xleftarrow{\ell} n_2, n_2 \in \mathbb{N}_{i-1}(n)\}$$

In practice, since doing so for every node in the database is expensive in terms of space, we employ an implementation similar to the idea of the inverted indexes. We use an index structure that for every label and for every distance can provide a list of all the nodes that have a label ℓ at the respective distance, and the cardinality of such labels.

Note that, once computed for each label ℓ and each $i \leq d$, W compactly represents the neighborhood of a node. For this reason, if we compute W for the nodes of the user sample as well, we can compare nodes in the database and nodes in the user sample, in order to know in advance which nodes can be pruned. We denote the d-neighborhood of a node n_s of graph S by $\mathbb{N}_d^S(n_s)$. A node $n \in N$ of $D = \langle N, E\rangle$ matches a node $n_s \in N_S$ in the user sample, and therefore is not pruned, if for each label l and a distance $i \leq d$, $|W_{n,\ell,i}| \geq |W_{n_s,\ell,i}|$ (ref. to Theorem 1 for a formal proof).

Using the ability to compare nodes through the compact representation of their neighborhood, we devise a way of fast eliminating pairs of the user sample and database nodes, respectively, that are unlikely to participate in an isomorphism match. Traditional techniques that compute isomorphisms compute matches of the different nodes independently and then try to combine the results.

We believe that this process can be optimized further, if the comparison of the nodes takes into consideration the previously computed matches. To implement this idea we adopt the notion of *simulation* [27]. A graph simulates another graph if it exists a way to map each transition on the first graph with a transition in the second.

Definition 6 (Simulation)**.** *Let $G_1 = \langle N_1, E_1\rangle$ and $G_2 = \langle N_2, E_2\rangle$ be two graphs.* G_2 simulates G_1 *if there exists a relation $\mathcal{R}$, such that, for every node $n_1 \in N_1$ and $n_2 \in N_2$ for which $(n_1, n_2) \in \mathcal{R}$ and $n_1 \xrightarrow{\ell} n_1'$, there exists a n_2' such that $n_2 \xrightarrow{\ell} n_2'$ and $(n_1', n_2') \in \mathcal{R}$*

Deciding whether one graph simulates another graph is known to be solvable in polynomial time with respect to the size of the graph [16]. The main idea of our approach is to perform multiple simulations of the user sample on the database graph while pruning the non matching nodes iteratively. The algorithm works as follows. First, it calculates the d-neighborhood for each node of the user sample. Then, a user sample node is selected as starting node. Although any node is a valid starting node we propose to pick the node with the lowest selectivity among the user sample nodes, with the hope to reduce the number of candidate matches between the user sample and database nodes. The selectivity is an estimate of the number of possible matches generated from a user sample node. The idea is to consider the number of adjacent nodes of a user sample node and the frequency of the labels of the edges connected to it. The selectivity of a node n is

$$Sel(n) = \ freq(n) + \sum_{i=1}^{d} \frac{1}{i} \sum_{W_{n,\ell,i}} |E^{\ell}|, \qquad (1)$$

where the frequency $freq(n)$ of a node n is defined as the sum of the number of outgoing and incoming edges. Similarly, we define the frequency of a label ℓ as the number of edges in the graph having label ℓ and we denote it as $|E^{\ell}|$. The less probable the combination of labels at a certain distance is, the lower the selectivity and the higher is the expected pruning power.

After having selected the starting node n_{min}, the algorithm retrieves the nodes in the database that match the node n_{min} and marks them as candidate mappings $\mu(n_{min})$, where $\mu \subseteq N_S \times N$ is the mapping between user sample and database nodes that the algorithm will compute. Then the algorithm iteratively checks, for each user sample node n_s not yet visited, that each adjacent edge of n_s matches the edges adjacent to the nodes $n \in \mu(n_s)$, verifying the label and the direction of the edge. If it does not match, then n is removed from $\mu(n_s)$, otherwise we consider a node n_1 adjacent to n a candidate for the user sample node n_s' adjacent to n_s, i.e., we insert it into $\mu(n_s)$, if the condition described by Theorem 1 holds. Finally, the user sample node n_s is marked as visited and removed from the candidate list. The steps of the algorithm are described in pseudo-code in Algorithm 2.

The following theorem guarantees that Algorithm 2 does not falsely discard any node while traversing the user sample nodes. However, it may introduce false positives, i.e., nodes that match the user sample nodes but are not included in an isomorphism.

Theorem 1. *Given a database $D = \langle N, E\rangle$ and a user sample S, let $\mathbb{N}_d$ and $\mathbb{N}_d^S$ be the d-neighborhood of D and S respectively. If there exists a subgraph-isomorphism $\mu : N_S \to N$, then $\forall n_s \in N_S, \mathbb{N}_d^S(n_s) \subseteq \mathbb{N}_d(n), n \in N, n \in \mu(n_s)$*

Proof. (by contradiction) Suppose that $(n_s, n) \in \mu$, but $\mathbb{N}_d^S(n_s) \not\subseteq \mathbb{N}_d(n)$, then there exists $i, 1 \leq i \leq v$ and a label ℓ such that $|W_{n_s,\ell,i}| > |W_{n,\ell,i}|$. For this reason we can say that there exists $n_s' \in W_{n_s,\ell,i}$, connected to $n_s'' \in N_{i-1}(n_s)$ by ℓ, i.e., $n_s' \xrightarrow{\ell} n_s''$. The latter assumption is true since we assume that μ is a subgraph-isomorphism. However, there does not exist any $\mu(n_s') \xrightarrow{\ell} \mu(n_s'')$, which contradicts the subgraph-isomorphism hypothesis. □

Additionally, a guarantee that the algorithm correctly computes multiple simulations of the user sample S, is offered by the following theorem.

Theorem 2. *Given a user sample S, if Algorithm 2 terminates with a complete exploration of the nodes S, then there exists in μ a simulation $\mathcal{R}$ of the user sample S.*

(The proof is omitted due to space limitations)

In the worst case, Algorithm 2 will have to traverse the entire database for each node. Thus, the complexity of the algorithm is $O(|N| * (|N_S| + |E_S|))$. Since the user sample is typically very small, the algorithm is, for the majority of practical cases, linear to the size of the graph. In implementation, to reduce the time computation of μ we used a hash map for storing the nodes of the user sample and their partial mappings.

The set of candidate mappings computed by Algorithm 2 is used to eliminate those nodes of the database that will never participate in an isomorphism with the user sample nodes. The reduced-size database that results from this elimination can then be fed to the XQ Algorithm of Section 4. We refer to this new algorithm as FastXQ.

6. AN APPROXIMATE SOLUTION

In the previous section, we described an exact solution to prune the search space, removing nodes that cannot possibly match the user sample. In this section, we propose an additional method that removes in advance nodes that are likely to not be relevant for the user, which we call ApFastXQ.

We aim at restricting in advance the search space in order to search for solutions only in the portion of the graph that is more likely to contain relevant answers. As already mentioned in the previous sections, both pairs *Yahoo!-Flickr* and *GM-Opel* are part of the solution space, but the pair *Yahoo!-Flickr* pair is more relevant to the user, and therefore we would like to restrict our search only to the subgraph that is containing the second but not the first.

In the following, we describe how we model this portion of the graph, which we call *Relevant Neighborhood* (Section 6.1). That portion is the subset of nodes with higher proximity to the nodes of the user sample. The intuition behind this is that nodes in the graph that are located far from the user sample will be also semantically distant from the user's intention as expressed in the exemplar query.

We model a *relatedness* measure based on the distance in the graph, and we use it to prune away nodes that are far away from the user sample before even looking for isomorphic structures.

It is clear that, while the approach described in the previous section is exact (does not discard any valid answers), this second approach is approximate: some correct answers could potentially be filtered out as they fall out of the *Relevant Neighborhood*. For this reason, we propose a principled way of measuring the *relatedness* and for pruning the graph, aimed at discarding only irrelevant solutions. We implement a function that iteratively retrieves the *Relevant Neighborhood* without traversing the entire graph (Section 6.2). As we show later (Section 8), thanks to the RelevantNeighborhood algorithm, by operating in this special portion of the graph, we can effectively reduce the search space. The restricted search space can then be given as input to Algorithm 1, without sacrificing the quality of the results. We can still apply on this subgraph the pruning techniques presented in the previous section and then look for isomorphic structures on a much smaller database. Hence, the ApFastXQ algorithm first applies the RelevantNeighborhood algorithm and then FastXQ.

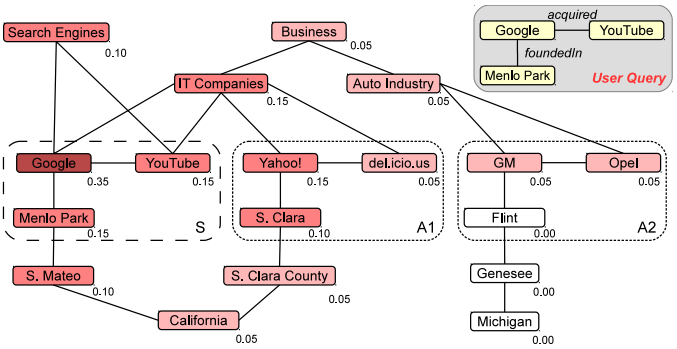

Figure 2: A visualization of APPV

6.1 Identifying the Relevant Neighborhood

To find the subset $\mathfrak{Q}_\rho$ of the set of answers $\mathfrak{Q}$, that contains the answers that are relevant to the user, we assume to have some measure of *relevance* ρ. Since the only evidence of the user's intent is the input query Q and the corresponding user sample S we have found in the first step of the exemplar query evaluation, we can define the set of relevant answers as $\mathfrak{Q}_\rho = \{A' \in \mathfrak{Q} | \rho(A', S) > \tau\}$, with $\tau > 0$ being a minimum threshold.

Since $\mathfrak{Q}_\rho$ is clearly a set of subgraphs in D, we say that the set of solution $\mathfrak{Q}_\rho$ is contained in the subgraph $D_\rho \subseteq D$, which is any subgraph of D that contains all the members of $\mathfrak{Q}_\rho$, relevant answers, and none of the remaining irrelevant solutions in $\mathfrak{Q} \backslash \mathfrak{Q}_\rho$. For this reason we call D_ρ the *Relevant Neighborhood* of the sample S.

This portion of the graph, being a subgraph itself, is identified by the subset of *relevant nodes* $N_\rho \subseteq N$. Those are nodes whose relevance measure ρ is within the a threshold τ, i.e., $N_\rho = \{n \in N | \rho(n, S) > \tau\}$. Operationally, we first identify the set of relevant nodes N_ρ, and then with only those nodes, we easily construct the subgraph $D_\rho \subseteq D$.

In our solution, we implement ρ as a distance measure on the graph, such that it measures the distance of every node from the nodes of the sample N_S, and we keep only nodes that are within a certain distance threshold from the sample. In order to compute this distance we propose APPV, an extension of the Personalized PageRank Vector (PPV), designed to exploit the properties of our problem. This is implemented by the RelevantNeighborhood algorithm.

6.2 The RelevantNeighborhood Algorithm

Our solution models the computation of the Personalized PageRank vector (PPV) [18] which is used as an estimate of the distances of the nodes in the graph from the subset of nodes in the user sample. In the literature Personalized PageRank [15, 18] is a well known technique that computes the PageRank biased towards the preferences of the user. In our case, user preferences are expressed through the query Q and for this reason we initialize the preference vector according to the nodes in the user sample S, which models the query Q in the database.

The main difference between the original PPV model and our solution, the Adaptive Personalized PageRank Vector (APPV), lays on the semantic of edges. Traditionally, edges between nodes are treated equally as they usually represent

Algorithm 3 RelevantNeighborhood

Input: User Sample $S : \langle N_S, E_S \rangle$
Input: Database $D : \langle N, E \rangle$
Input: Teleportation probability c
Input: Threshold τ
Output: Subgraph $D' \subseteq D$

```
1: Ā ← AdjacencyNormalized(D, S)
2: p ← [0] × N
3: for each q_i ∈ N_S do
4:     p[q_i] ← 1/|N_S|
5: v ← ComputeAPPV(A, p, c, τ)
6: N_D' ← Nearest(N, v)
7: D' ← GetSubgraph(D, N_D')
8: return D'
```

just a link from one webpage to another (i.e., they are of the same kind). In contrast, our model adapts to the various edges and their labels, according to S. In particular, the edges in our model may represent different kinds of relationships. It is therefore natural to differentiate based on the information carried by different edges, as some relationships are more informative than others. Moreover, labels that do appear in the user query should be treated differently when computing the PageRank, because they represent the user preference.

Figure 2 depicts the output of the computation on the graph of our running example. Here all the nodes have been assigned the weights from the final APPV, computed using the set of nodes in the sample as initial preferences.

The set N_ρ, which satisfies the selectivity requirement, consists of nodes with Personalized PageRank score higher than a minimum threshold τ, $0 < \tau < 1$.

This whole process, presented in Algorithm 3, returns the portion of the graph that is combined with Algorithm 2 to produce a restricted database D' which is provided to Algorithm 1 instead of D.

Assume a model of the database $D = \langle N, E \rangle$, and let A^D be the adjacency matrix of this graph. If $|N|$ is the number of nodes in the database, then A^D is an $|N| \times |N|$ square matrix. In this matrix, we have that $0 < A^D_{ij} \leq 1$ if and only if the node i has a relationship e^ℓ_{ij} with node j with label ℓ; otherwise, we have $A^D_{ij} = 0$. In this way, the element A^D_{ij} models the amount of information that is transferred from node i to node j by the edge e^ℓ_{ij} as a function of its label ℓ. In our solution, the values in A^D are proportional to the amount of information [29] carried by the edge e^ℓ_{ij}, which is:

$$I(e^\ell_{ij}) = I(\ell) = \log \frac{1}{P(\ell)} = -\log P(\ell) \tag{2}$$

$$P(\ell) = \frac{|E^\ell|}{|E|} \tag{3}$$

where E^ℓ is the set of edges with label ℓ. Note that the frequency of a label can be easily computed in the database.

In order to account for the importance of the edges in the user sample, we additionally define matrix A^S, which is constructed from the adjacency matrix of the user sample S, but with additional columns for all the other nodes in the database, and the values of these additional columns/rows padded with zeros. In other words, we construct an $|N| \times |N|$ square matrix with $0 < A^S_{ij} \leq 1$ if the nodes i and j are connected by an edge and that edge is part of the user sample S, and with A^S_{ij}=0 otherwise.

We then combine the two matrices into the matrix $\bar{A} = A^D + A^S$ and normalize it. Under this transformation $\bar{A}$ becomes the transition probability matrix for the knowledge-base graph, where more relevance is given to edges carrying more information, as well as to edges with labels that appear in the query. We also define $\boldsymbol{p}$, an $|N| \times 1$ column vector, which serves as the normalized preference vector for which $\boldsymbol{p}[i] \neq 0$ *iff* $n_i \in N_S$, i.e., $0 < \boldsymbol{p}[i] \leq 1$ if and only if the node i is in S. Given the column normalized transition probability matrix $\bar{A}$, the teleportation probability c, and the preference vector $\boldsymbol{p}$, our technique adheres to the Personalized PageRank semantics [10, 18].

Thus, the APPV $\boldsymbol{v}$ is defined as the stationary distribution of the Markov chain with state transition given by the matrix

$$(1-c)\bar{A}\boldsymbol{v} + c\boldsymbol{p} \tag{4}$$

where the *teleportation* probability $c \in (0, 1)$ is typically ≈ 0.15, with small changes in this value having little effect in practice [26].

The exact computation of this vector typically requires $O(n^2)$ time and space. Performing the computation through power iteration requires $O(nt)$ time, where t is the number of iterations to be performed. Nevertheless, this computation is still not practical for very large graphs.

In order to compute this value fast, we extend the template proposed in [3] and apply an approach similar to the *weighted particle filtering procedure* proposed in [21] but extended to correctly take into account the *teleportation probability*, and to consider the non-uniform edge weights that we previously introduced. The extension is shown in Algorithm 4.

Algorithm 4 ComputeAPPV

Input: Database D
Input: Node vector $\boldsymbol{p}$
Input: restart probability c
Input: threshold τ
Output: Approximate APPV $\boldsymbol{v}$

```
1: for each q_i ∈ p do
2:     p[q_i] ← p[q_i] × 1/τ
3: v ← p
4: while ∃ n_i ∈ p | p[n_i] ≠ 0 do
5:     aux ← [0]
6:     for each n_i ∈ p | p[n_i] ≠ 0 do
7:         particles ← p[n_i] × (1 − c)
8:         for each n_i → n_j ∈ D (Sort by Ā_ij Desc.) do
9:             if particles ≤ τ then
10:                break
11:            passing ← particles × Ā_ij
12:            if passing ≤ τ then
13:                passing ← τ
14:            aux[n_j] ← aux[n_j] + passing
15:            particles ← particles − passing
16:    p ← aux
17:    for each n_i ∈ p do
18:        v[n_i] ← v[n_i] + p[n_i]
19: return v
```

2) starting from each node with a non-zero value in $\boldsymbol{p}$. At each iteration (lines 6-15), they split among the neighbors of the node they are currently visiting, but we prevent them to split to arbitrarily small sizes, limiting them to have minimum size τ (lines 12-13). When spreading the particles among the neighbors, the algorithm gives preference to the edges with higher weights. The restart probability c will dissipate part of the particles at every iteration (line 7), and the algorithm will stop when no more particles are floating around.

At the end of the algorithm, we return the APPV containing the scores that have been accumulated through each iteration on every node. We then keep the subset of the graph containing only those nodes with a score higher than some threshold and the edges connected to them (line 6-7 in Algorithm 3). Since we are dealing with an iterative approximation, we will take all nodes which have been visited by at least one particle, which actually means that we set the threshold to be equal to τ.

7. RANKING QUERY ANSWERS

Once the answers have been computed from the user sample, they need to be ranked in order to either be returned sorted to the user that posed the query, or to select only the k most promising candidates, i.e., the top-k. To do this, we introduce a novel ranking function that is a linear combination of two scores, namely, the structural similarity score $\mathcal{S}$ based on the d-neighborhood and the amount of information as provided by the Personalized PageRank, which indicates the importance of a label in the graph. The score of each answer is computed by using the above two parameters to compare the answer to the user sample.

Most node similarity measures proposed in the literature are based on the concept of graph similarity and isomorphism. This is the case for Graph Edit Distance [13], which is computed with a reduction to graph isomorphism, and is therefore inapplicable to our problem, due to its high time complexity. A different method is proposed in [19] and is based on a vectorial representation of nodes. This idea seems suitable for our settings, thus we extended it in order to capture the differences among nodes that emerge when taking into account the edge-labels of the neighbors. We also embed distance information aiming at giving different weights to nodes based on their distance from the sample. Thus, for every node n we build a vector containing a value for every label $\ell \in L$ in the graph, and we compute this score as

$$\sigma(n, \ell) = \sum_{i=1}^{d} \frac{I(\ell)|W_{n,\ell,i}|}{i^2} \tag{5}$$

Given the vectorial representation of two nodes, we compute the node similarity $\mathcal{S}$ using a metric for vectors, such as the Jaccard, euclidean distance or cosine similarity. Note that our vectorial representation contains already the computed score σ. In our experiments we use cosine similarity but any other similarity metrics can also be used. Therefore, the *structural similarity* between a node n_s of the user sample and n is computed as follows:

$$\mathcal{S}(n_s, n) = \frac{\sum_{\ell}^{L} \sigma(n_s, \ell)\sigma(n, \ell)}{\sqrt{\sum_{\ell}^{L} \sigma(n_s, \ell)^2}\sqrt{\sum_{\ell}^{L} \sigma(n, \ell)^2}} \tag{6}$$

The structural similarity above does not take into account the proximity measure of the results with respect to the user sample. Therefore, we consider a linear combination, parametrized by λ, between the node similarity (structural) and the Personalized PageRank (proximity) as follows.

$$\rho(n_s, n) = \lambda\mathcal{S}(n_s, n) + (1 - \lambda)\boldsymbol{p}[n]. \tag{7}$$

where $\boldsymbol{p}[n]$ is the APPV as defined in the previous section.

The final score of a query answer with respect to the user sample S is the sum of $\rho(n_s, n)$ for every node n_s in the user sample with respect to the corresponding (isomorphic) node n in the query answer. Note that the choice of λ is data dependent. A value λ close to 1 favors results that are mostly similar to the neighbors of the user sample nodes. On the other hand, a value close to 0 will take into account only solutions that are close to the original query. For this reason, we can see λ as a diversification parameter that depends on the user and on the data. This is also the approach taken by most diversification models [1].

8. EXPERIMENTAL EVALUATION

In this section, we experimentally validate our solution by comparing it to other approaches, and measuring its performance.

Queries: We extracted 90 real queries from the AOL query log to use as the query test set, and manually mapped them to the knowledge base[2]. These queries are highly heterogeneous in terms of size and frequency of the edge labels. We then used 50 of these queries for the user study.

Datasets: We downloaded a full dump of the Freebase knowledge base[3] in August 2012. We removed every triple that is used as internal specification for the community (such as the user and groups data and discussion topics) obtaining a fully connected graph of 53 million nodes and 213 million edges (occupying 24GB of main memory). To the best of our knowledge this is the biggest graph used in this context in the literature, and the first time that the entire Freebase graph is used for this purpose. While related works [19, 33, 31] evaluate their methods on just a small part of Freebase, we explored solutions that scale to the full size of the knowledge-base. In the following we refer to it as `Real`. Based on `Real` we generated 10 synthetic datasets embedding 20 samples of the test set in different points of the graph. We performed a breadth first traversal of the graph from a fixed starting node and we randomly chose to embed an answer according to a distribution that decreases exponentially with the distance from the starting node (thus modeling answers at varying distances). Only a fixed number of answers are embedded in the graph. To test the scalability with respect to the graph size, we generated graphs having 0.5M, 1M, 5M, 10M and 20M nodes, and 1K embedded queries. We denote them as `GSize-x` where `x` is the graph size. Similarly, to test the scalability with respect to the number of answers to retrieve, we generated graphs with 10M nodes, and 0.5K, 1K, 2K, 5K and 10K embedded answers. We denote them as `QSize-x` where `x` is the number of generated answer.

Experimental Setup: Due to space limitations, we do not report the experiments with varying the parameter d of It-

[2]List of queries: `http://disi.unitn.eu/~themis/exemplarquery`
[3]`http://download.freebase.com/datadumps/`

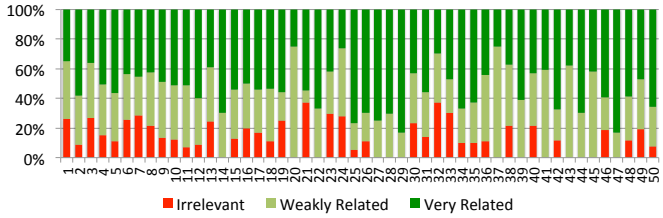

Figure 3: Percentage of relevant and irrelevant results per query

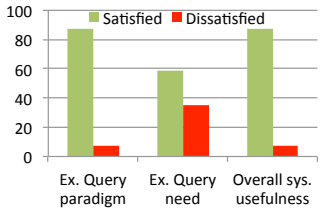

Figure 4: Percentage of satisfied and dissatisfied users

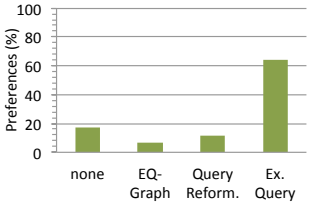

Figure 5: Comparison of methods applied to the Exemplar Query task

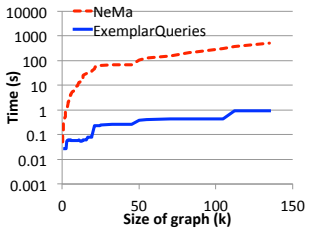

Figure 6: Time vs Size of graph with NeMa and our approach (Real dataset).

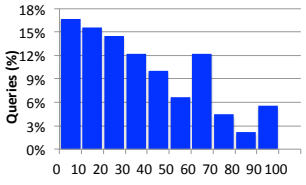

Figure 7: Pruning time gain distribution (Real dataset).

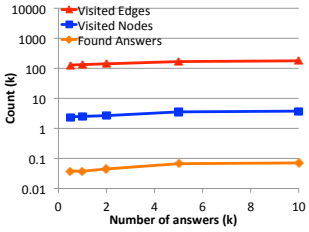

Figure 8: Count vs number of answers (QSize-x dataset).

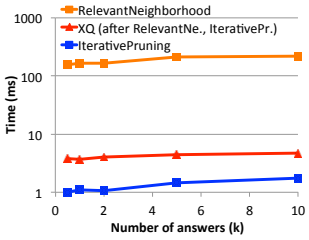

Figure 9: Time vs number of answers (QSize-x dataset).

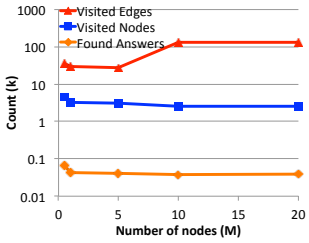

Figure 10: Count vs number of nodes (GSize-x dataset).

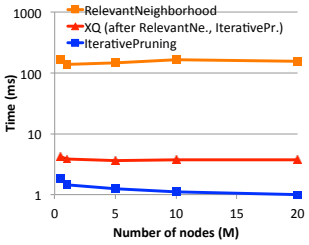

Figure 11: Time vs number of nodes (GSize-x dataset).

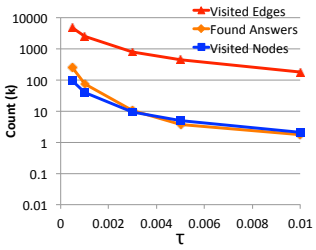

Figure 12: Count vs ApFastXQ threshold (Real dataset).

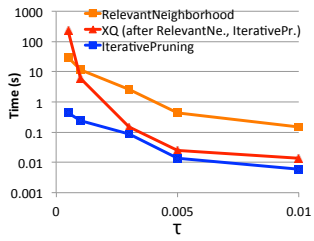

Figure 13: Time vs ApFastXQ threshold (Real dataset).

erativePruning (Section 5). For the rest of this paper, we use $d = 3$ since it proved to be a reasonable choice and allowed us to compute answers in less than 1 second. We also observed that $\lambda = 0.3$ (see Section 7) is a good compromise for retrieving diverse and qualitative results. In Section 8.4 we study the effect of varying the threshold parameter τ (see Section 6.2), for which the default value is 0.003. All the reported results are averages over 5 consecutive runs. We implemented our solution in Java 1.6, and ran the experiments on a $i686$ Intel Xeon $X3220$ 2.40GH, 32Gb RAM machine over Linux kernel $v2.6.30$. The graphs are loaded into main memory using our graph library available under open source license[4].

Implemented Algorithms: Apart from FastXQ and ApFastXQ, we implemented three additional algorithms from related works:

QueryReformulation: An algorithm that produces query reformulations by mining sessions from query logs in a term-level fashion [32]. The model is trained on the AOL query log and the suggestions are based on our query test set.

EQ-Graph: Entity-query graph is a model that computes serendipitous suggestions starting from entity mentions in a page [9]. We manually adapted our query test set to work in this setting associating to each node the corresponding Wikipedia page (or the best Wikipedia page that represents the node). The model is trained on a big query log from the Yahoo! Search Engine.

NeMa: This algorithm [20], and other previous works, are based on the assumption that there exists a truly small set of correct answers to a graph query. In our case, this property does not hold: what we need to find is a generalization of the original sample already present in the dataset. Therefore, we implement their technique taking into account edge label matches instead of node matches. The authors kindly provided us a C++ implementation (compiled using gcc $v4.4.3$).

Summary of Experiments: We set up user studies, where we evaluated the usefulness (Section 8.1) of our approach, and also compared it (Section 8.2) to the related works we discussed above.

[4] https://github.com/mutandon/Grava

Q1: Google - YouTube - Menlo Park
Google - YouTube - Menlo Park
Yahoo! - LAUNCH Media - Stanford University
Yahoo! - Musicmatch - Stanford University
Yahoo! - Right Media - Stanford University
Yahoo! - Inktomi Corporation - Stanford University

Table 1: Top-5 results with NeMa for "Google YouTube Menlo Park"

Q2: Condom - Sex - HIV infection
Water purification - Fecal-oral route - Cholera
Smoking cessation - Vector - Diabetes mellitus
Oral Transm. - Cytomegalovirus Infections - Oral Transm.
Oral Transmission - Cerebral palsy - Cytomegalovirus
Water purification - Fecal-oral route - Cholera

Table 2: Top-5 results with NeMa for "Condom Sex HIV infection"

Q1: Google - YouTube - Menlo Park
Google - AdMob - Menlo Park
Google - DoubleClick - Menlo Park
Yahoo! - del.icio.us - Santa Clara
Microsoft - Powerset - Albuquerque
A&E Television - Lifetime Ent. Services

Table 3: Results for exemplar query "Google YouTube Menlo Park"

Q2: Condom - Sex - HIV infection
Safe sex - Sex - HIV infection
Sexual abstinence - Sex - HIV infection
Safe sex - Vertical transmission - Hepatitis B
Safe sex - Vertical transmission - Syphilis
Hand washing - Droplet Contact - Cold

Table 4: Results for exemplar query "Condom Sex HIV infection"

Section 8.2 further studies the performance and quality of our solutions when compared to a query answering system for graphs. In Section 8.3, we study the impact of IterativePruning on query time, and the correlation between pruning time and node selectivity. Section 8.4 presents the effect of τ on query time. Finally, scalability experiments of the proposed algorithms are described in Section 8.5.

Summary of Results: Our user studies demonstrate that 92% of the users exemplar queries are relevant and useful for search tasks, and that existing approaches are not able to provide effective solutions for our problem, while our method identifies meaningful results with 81% precision. When compared to existing approaches, our algorithm retrieves results almost 3 orders of magnitude faster than a state of the art graph matching algorithm. We observe that the proposed pruning strategy reduces the query-time by 30% on average, with even higher improvements when we choose starting nodes with low selectivity. Finally, the last set of results demonstrate the scalability of our approach to the largest knowledge graph available in the field (53M nodes, 213M edges), while maintaining low response time.

8.1 Usefulness

In order to assess the quality of the proposed solution, we conducted the following user study. We asked 94 users (uniformly distributed with respect to education level, age and country) to evaluate our system. For each query in the test set, we provided an explanation of the topic, the query intention, and our answer set with the top-10 results provided by our ranking function. We asked each user to rate each result as *irrelevant*, *weakly related*, or *very related* with respect to the topic and the expressed query intention. Each user evaluated between 2 to 10 queries (on average 8). The users provided 4540 marks in total, shown in Figure 3: 81% of our results are marked as relevant (weakly or strongly) and only 19% of them are not considered relevant suggestions. Out of the 427 suggestions we produced, 172 (i.e., 40%) are judged highly relevant by more than 50% of the users. Note also that each exemplar query contains at least one relevant (weakly or strongly) result for 99% of the users. Moreover, we ask each user to express her opinion with respect to (a) the idea of using examples as a search paradigm, (b) whether she already had the need of searching using exemplar queries, and (c) the usefulness of the system in general. As shown in Figure 4, 92% of the users considers the exemplar queries paradigm and the overall system useful for retrieving additional and relevant information. Moreover, 62% of the people interviewed declared that they had already had the need to perform this kind of exemplar queries search in the past (but there was no system to support them).

8.2 Comparison to Previous Work

In the following we compare our method against two different approaches: (a) algorithms that produce related queries, and (b) an approximate query answering technique for graphs.

Related Queries: We implemented and compared with the methods QueryReformulation and EQ-Graph mentioned earlier, through a user study similar to the one presented in Section 8.1. For each query in the test set, we presented to users three groups of suggestions: one produced with our method, and one produced by each one of the two methods above. We then asked users which of the three groups of suggestions they considered the most helpful for each query task.

The results, depicted in Figure 5, show that in 64% of the cases the users preferred our solution to the other two. Furthermore, for 78% of the queries that received more than 2 marks, the majority of users preferred our solution. In 18% of the cases none of the proposed solutions were satisfying, neither the answers proposed by our model nor those produced by the other algorithms. Overall, the two competing approaches together was preferred by less than 30% of the users, none of them choosing the two approaches in all the queries.

Approximate Query Answering on Graphs: We now present the comparison between our approach and NeMa [20], a state of the art technique for answering approximate queries on graphs. Since on `Real` a single query takes NeMa more than 13 hours to process, we test NeMa on graphs obtained after applying RelevantNeighborhood on our query test set, thus giving it an advantage. The results in Figure 6 show that NeMa is almost three orders of magnitude slower than our algorithm. This suggests that a query answering technique for graphs is not applicable to our setting.

We also provide anecdotal evidence comparing the top-5

results from our method and NeMa. Tables 1 and 2 show the top-5 results of NeMa for two different exemplar queries compared with the results of our algorithm, shown in Tables 3 and 4 (for our algorithm, we report the top-2 results containing query terms, the top-2 results not containing query terms, and for reference, the lowest ranking result). We observe that if the structure of the exemplar query is complex (e.g., it contains cycles), NeMa fails to find the correct answers, mapping different query nodes on the same graph node as depicted in Table 2-row 3, where the same graph node,"Oral Transmission", is used twice. Actually, 87% of the answers produced by NeMa are not isomorphic to the test queries, producing results that contain the same node more than once, and thus, leading to poor results. Furthermore, the top answers proposed by NeMa for Q2 contain diseases that are not sexually transmitted (e.g., diabetes that is ranked 2nd), a situation that does not occur with our algorithm.

8.3 Pruning Effectiveness

In the next experiments we show the impact of pruning on query time and the effect of selectivity on pruning time.

Pruning impact: We perform a batch of experiments using the query test set, comparing the query time with and without applying IterativePruning, and depict the results in Figure 7. (We note that, as discussed in Section 5, our pruning technique does not modify the quality of the final result set, neither does it discard any relevant result.) Applying pruning results in 3% to 99% less query time. Moreover, we observe that for 17% of the queries, pruning does not affect the query time. We notice that pruning is more effective when the frequencies of the edge labels of the sample in the graph are high, since a large part of the graph is eliminated with fewer operations. This observation allows us to run the IterativePruning on demand. On average, IterativePruning reduces query time by 30% and the graph size by 80% (by removing non-matching edges). This entire batch of experiments takes 38 minutes to execute without pruning and 17 minutes with pruning, saving 55% of the total time.

Pruning Selectivity: We study the performance of pruning in terms of time as a function of the selectivity of the starting node in the sample. Remember that low selectivity means better pruning (see Equation 1). We run experiments measuring the correlation between time and selectivity, selecting the different nodes of the sample as starting nodes. The results show a positive correlation of 0.57 between selectivity and time performance, which is statistically significant at the 0.01 significance level. We conclude that starting from a low selective node positively impacts the pruning time, with savings up to 87%.

8.4 Calibrating RelevantNeighborhood

We study the effect of τ on RelevantNeighborhood in terms of time and quality of the results. Parameter τ of RelevantNeighborhood determines the degree of approximation of the estimation of PPV of each node. In Figure 12, we plot the size of the neighborhoods (counts of nodes and edges) visited for increasing values of τ (from 0.0005 to 0.01), and the number of answers retrieved in each case. The observed decrease in the number of visited nodes and edges is directly proportional to the number of answers retrieved, and exhibits an exponential decay as τ increases. Figure 13 shows that with larger values of τ the time needed to compute the results decreases in the same manner.

We now evaluate the quality of the answers produced by ApFastXQ, by measuring precision at 1,5,10,50,100, where precision at k (abbreviated $P@k$) is defined as the fraction of results produced by FastXQ that are also produced by ApFastXQ in the first k positions. Table 5 shows that overall precision is high, especially for the top positions. Any value of τ between 0.003 and 0.005 is a reasonable choice, leading to high precision and an average query time of less than 2.4 seconds. Evidently, the choice of this parameter depends on the application. In a biological setting, where precision is more important than time, $\tau = 0.002$ could be a reasonable choice, producing very precise answers in about 10 seconds. On the web, where timely answers are needed, $\tau = 0.005$ can still offer precise answers in the top positions, in less than 1 second. In our experiments, we used $\tau = 0.003$.

τ	$P@1$	$P@5$	$P@10$	$P@50$	$P@100$
0.002	1	0.99	0.99	0.85	0.75
0.003	1	0.97	0.94	0.80	0.73
0.004	1	0.95	0.93	0.71	0.60
0.005	1	0.94	0.92	0.66	0.56

Table 5: Precision of ApFastXQ varying τ

8.5 Scalability

We present the scalability experiments as a function of the number of answers and the size of the database. In Figure 8 we show the number of visited edges and nodes, as well as the number of results when the number of embedded answers increase (recall that `QSize-x` contains exactly x answers for each exemplar query). The time of ApFastXQ is the sum of the times shown in Figure 8. We observe that using RelevantNeighborhood as the number of answers increases from 60 to 100, the number of explored nodes remains almost the same. This behavior is expected, since RelevantNeighborhood does not explore more nodes as long as the structure of the graph remains almost unchanged, but it finds more answers embedded in the same subgraph.

Conversely, if the size of the dataset increases and the number of answers is fixed (Figures 10 and 11) it is less likely to find answers close to the exemplar query. As expected, since the number of nodes explored is almost the same (see Figure 10) the time is constant, even though we move from 500k to 20M nodes. This supports our design choice, since changes in the peripheral part of the graph do not affect the APPV algorithm.

9. RELATED WORK

Query Modification. Many different works study ways to provide the user with answers that may be of interest even if they were not explicitly requested in the query. *Query refinement* [22] extends the user query in order to retrieve more precise results [2, 7, 28, 32] using some external knowledge. In our work we are not trying to alter the query, but only use it as a sample that can lead us to additional queries generating resources of interest. *Query relaxation* [22, 23], on the other hand, relaxes an over-specified query that returns no answers to allow a non-empty answer set to be produced. Our approach is somehow similar, however, query relaxation is driven by the conditions in the query. It will

not include results that are similar to those the user query generates, unless they are satisfying a subset of these conditions. In contrast, our approach adds additional results by using similarities at the data level. *Related queries* deal with the discovery of queries generating results of possible interest to a user based on a query that the user has already posed. Their discovery is based on information like query logs [2, 32], document corpuses [7], knowledge bases [25] or wikis [9]. Since our work can also be used to suggest related queries as explained in the introduction, we can be seen as complementary to these approaches, offering a new way of generating related queries.

Another group of works that do not try to extend/improve the query results with new data, but only to organize them in some way that is more comprehensive to the user, is the one of *faceted-search* [12] and *query categorization* [30]. Despite the fundamental difference from our approach, these works are also aiming at increasing the user satisfaction.

Query Answering on Graphs. Most works on finding a graph structure in a large graph exploit graph edit distance that measures similarity between subgraphs [13]. Computing the graph edit-distance is NP-hard and numerous indexing and pruning techniques have been proposed to improve performance [31, 33]. Our solution is mainly related to works in which the representation of a portion of the neighborhood of a node is used to prune in advance nodes that are unrelated to those in the query [19, 20, 34].

10. CONCLUSIONS

In this paper we introduced a novel query paradigm called exemplar queries. We formally defined it and described how it is applied on a graph-based data model, where it requires search for graph-isomorphism in order to evaluate a query. We discussed why traditional query answering on graphs is not applicable in this context, and proposed an exact solution based on an effective and theoretically sound pruning technique, alongside an efficient approximation algorithm. Moreover, we proposed a novel pruning technique based on the concept of d-neighborhood and of bi-simulation. We evaluated the efficiency and effectiveness of our approach using one of the biggest multigraphs in the literature, and coupled our results with a user study that confirms the usefulness of the proposed system. As future work, we are considering to study the case in which we have multiple exemplar queries for the same desired answer set.

Acknowledgments: This work was partially supported by the Trento RISE Big Data project [4]. We would also like to thank the authors of [9] and [20] for making available their code to run our experiments.

References

[1] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *WSDM*, pages 5–14, 2009.

[2] A. Anagnostopoulos, L. Becchetti, C. Castillo, and A. Gionis. An optimization framework for query recommendation. In *WSDM*, pages 161–170, 2010.

[3] R. Baeza-Yates, P. Boldi, and C. Castillo. Generalizing pagerank: damping functions for link-based ranking algorithms. In *SIGIR*, pages 308–315, 2006.

[4] I. Bedini, B. Elser, and Y. Velegrakis. The trento big data platform for public administration and large companies: Use cases and opportunities. *PVLDB*, 6(11):1166–1167, 2013.

[5] S. Bergamaschi, E. Domnori, F. Guerra, R. Trillo Lado, and Y. Velegrakis. Keyword search over relational databases: a metadata approach. In *SIGMOD*, pages 565–576, 2011.

[6] S. Bergamaschi, F. Guerra, S. Rota, and Y. Velegrakis. A hidden markov model approach to keyword-based search over relational databases. In *ER*, pages 411–420, 2011.

[7] S. Bhatia, D. Majumdar, and P. Mitra. Query suggestions in the absence of query logs. In *SIGIR*, pages 795–804.

[8] P. Boldi, F. Bonchi, C. Castillo, and S. Vigna. Query reformulation mining: models, patterns, and applications. *IR*, 14(3):257–289, 2011.

[9] I. Bordino, G. De Francisci Morales, I. Weber, and F. Bonchi. From machu_picchu to rafting the urubamba river: anticipating information needs via the entity-query graph. In *WSDM*, pages 275–284, 2013.

[10] S. Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. In *WWW*, pages 571–580, 2007.

[11] X. Dong, A. Y. Halevy, and J. Madhavan. Reference Reconciliation in Complex Information Spaces. In *SIGMOD*, pages 85–96, 2005.

[12] Z. Dou, S. Hu, Y. Luo, R. Song, and J. Wen. Finding dimensions for queries. In *CIKM*, pages 1311–1320, 2011.

[13] X. Gao, B. Xiao, D. Tao, and X. Li. A survey of graph edit distance. *Pattern Analysis and applications*, 13(1):113–129, 2010.

[14] S. Gauch and J. B. Smith. Search improvement via automatic query reformulation. *TOIS*, 9(3):249–280, 1991.

[15] T. H. Haveliwala. Topic-sensitive pagerank. In *WWW*, pages 517–526, 2002.

[16] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *FOCS*, pages 453–462. IEEE, 1995.

[17] B. Jansen, D. Booth, and A. Spink. Determining the informational, navigational, and transactional intent of web queries. *Information Processing & Management*, 44(3):1251–1266, 2008.

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

[19] A. Khan, N. Li, X. Yan, Z. Guan, S. Chakraborty, and S. Tao. Neighborhood based fast graph search in large networks. In *SIGMOD*, pages 901–912, 2011.

[20] A. Khan, Y. Wu, C. C. Aggarwal, and X. Yan. Nema: Fast graph search with label similarity. In *PVLDB*, pages 181–192, 2013.

[21] N. Lao and W. W. Cohen. Fast query execution for retrieval models based on path-constrained random walks. In *KDD*, pages 881–888, 2010.

[22] C. Mishra and N. Koudas. Interactive query refinement. In *EDBT*, pages 862–873, 2009.

[23] 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):1762–1773, 2013.

[24] D. Mottin, T. Palpanas, and Y. Velegrakis. Entity Ranking Using Click-Log Information. *IDA Journal*, 17(5):837–856, 2013.

[25] V. M. Ngo and T. H. Cao. Ontology-based query expansion with latently related named entities for semantic text search. In *IJIIDS*, volume 283, pages 41–52. 2010.

[26] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999.

[27] D. Park. *Concurrency and automata on infinite sequences*. Springer, 1981.

[28] Y. Qiu and H.-P. Frei. Concept based query expansion. In *SIGIR*, pages 160–169, 1993.

[29] C. E. Shannon. A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(1):3–55, Jan. 2001.

[30] D. Vallet and H. Zaragoza. Inferring the most important types of a query: a semantic approach. In *SIGIR*, pages 857–858, 2008.

[31] X. Wang, X. Ding, A. K. H. Tung, S. Ying, and H. Jin. An efficient graph indexing method. In *ICDE*, pages 210–221, 2012.

[32] X. Wang and C. Zhai. Mining term association patterns from search logs for effective query reformulation. In *CIKM*, pages 479–488, 2008.

[33] X. Yan, P. S. Yu, and J. Han. Graph indexing: a frequent structure-based approach. In *SIGMOD*, pages 335–346, 2004.

[34] P. Zhao and J. Han. On graph query optimization in large networks. *VLDB J.*, 3(1-2):340–351, 2010.

[35] M. M. Zloof. Query by example. In *AFIPS NCC*, pages 431–438, 1975.