Multi-Example Search in Rich Information Graphs

Matteo Lissandrini[◇], Davide Mottin[§], Themis Palpanas[⋆], Yannis Velegrakis[◇]

[◇] *University of Trento,* [§] *Hasso Plattner Institute,* [⋆] *Paris Descartes University*

[◇]{ml,velgias}@disi.unitn.eu, [§] davide.mottin@hpi.de, [⋆] themis@mi.parisdescartes.fr

***Abstract*—In rich information spaces, it is often hard for users to formally specify the characteristics of the desired answers, either due to the complexity of the schema or of the query language, or even because they do not know exactly what they are looking for. Exemplar queries constitute a query paradigm that overcomes those problems, by allowing users to provide examples of the elements of interest in place of the query specification. In this paper, we propose a general approach where the user-provided example can comprise several partial specification fragments, where each fragment describes only one part of the desired result. We provide a formal definition of the problem, which generalizes existing formulations for both the relational and the graph model. We then describe exact algorithms for its solution for the case of information graphs, as well as top-k algorithms. Experiments on large real datasets demonstrate the effectiveness and efficiency of the proposed approach.**

I. Introduction

We are witnessing a great deal of work towards novel query paradigms that better support both expert and non-expert users in coping the increasing complexity of data structures and schemas [1], [2]. User friendliness, language independence, and lack of full schema awareness have become fundamental factors in these efforts. There are many real-world situations, where such paradigms found applications, including the exploration of complex big data collections like open data [3].

By-example methods have become a popular paradigm in that category. They aim at simplifying information access by facilitating the specification process of the user's need [4], [5], [6], [7], [8], [9], [10], [11], [12]. They do so by having the user to provide an example of the elements of interest instead of a query and letting the system infer the conditions that such elements should satisfy. Existing works in relational databases expect the user to present some partially specified tuples that should be contained in the desired result-set [13], provide examples that are marked as relevant or irrelevant [7], specify tuples alongside explanations [8], or desired entities [9]. For graph-data, the user is expected to provide as an example a subgraph that is part of the desired result set [6], [10]. There the example that the user provides does not contain only information about components of interest, but also information on how these components are connected.

A limitation of the aforementioned approaches is that they assume there exists one example (structure or template) that describes the user needs and they expect that such example is known by the user. This is not always the case. Some relational approaches have allowed for incomplete examples [13], but partial examples in the case of graphs have not been studied so far [6], [10]. This becomes a limitation since users often are not aware of a single example that fully characterize what they are looking for. Previous work for relational databases [13] have shown that in several domains it is often easier to provide multiple partial examples, and expect the system to infer the complete specification by combining the information from the many examples.

The importance of combining and connecting distinct examples in the context of a single query has also been argued in a recent study on partial topology-based network search [14], which focuses on finding the connections between node-label isomorphic structures in an undirected graph. The provided exact solution, unfortunately, does not scale to large graphs [14].

Searching by providing multiple examples finds application in many different real-world scenarios. When, for example, lawyers are searching for similar court cases that involve the combination of more than one complex infringement, each example can refer to details of prior judgments and results are other judgments where those infringements appeared together. For biologists, the ability to provide multiple examples facilitates the search for complex molecular structures that contain certain substructures of interest. As a third example, advertisers could use friends, posts, and products from a network of existing customers to identify the target audience for their campaign. In all the above scenarios, it is not possible for one to come up with a single example describing all the desired specification, but even if this was possible, the different ways that these specifications can be combined are never considered from the existing approaches. The conjunction of the specifications is the unique and default option that has been considered.

In this work, we propose a novel method for query answering that is based on the ability to identify elements that were not known to the user, but share properties with some user-provided examples. We refer to this kind of queries as *multi-exemplar* queries, to emphasize its nature as an extension of previous works on example-driven query paradigms [6], [10], that required unique, complete and complex examples as inputs. We focus specifically on the case of large complex graphs. We assume that the examples that the user provides are in the form of a graph,

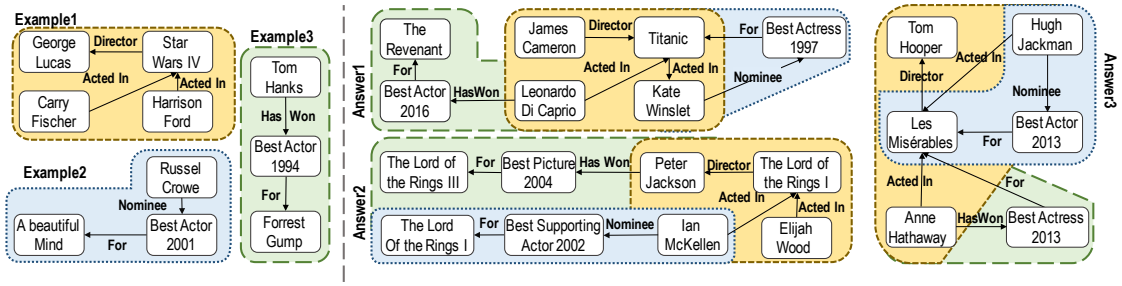

Fig. 1. Multiple examples, and some of the possible answers proposed with multi-exemplar query semantics.

and we look to find similar cases within a large knowledge graph. Once these cases have been identified, parts of them will have to be combined to form the final answers.

There are different challenges in performing the above tasks successfully to implement a multi-exemplar query mechanism. First, after the provided examples have been identified in the knowledge base, combining parts of them to form the final answers is a combinatorial problem that requires similar structure identification. In the case of graphs, similar structures are often identified through isomorphic structure discovery [6], [10], [15]. This makes the task exponential since all the possible combinations of all the structures similar to those provided by the user will have to be considered. Finally, the very nature of the graph data and their size poses some additional performance challenges. In certain cases, intermediate results may reach the tens of millions of graphs when the technique is applied to some real worlds knowledge bases.

An Illustrative Example. Consider a movie aficionado consulting an online resource, such as the Google Knowledge Graph[1], for movies where directors and actors have been nominated for, or won a prize. She is aware of some examples that describe her interests (Figure 1, left). She knows about *George Lucas* directing *Star Wars*, with actors *Carry Fischer* and *Harrison Ford*. She recalls *Russel Crowe* being nominated for the *Best Actor Academy award* for his role in *A beautiful mind*. Moreover, she also remembers about *Tom Hanks* winning *as Best Actor* with *Forrest Gump*.

Note that none of these examples by itself includes all the information she is looking for. Although she could perform a separate search for each one of them, and then manually compare the three different result-sets to come up with a list of movies, directors and actors with all the required information, this would require an unacceptable amount of work.

Instead, with a multi-exemplar query semantics, she could provide all three as input, and the system would retrieve various different answers (Figure 1, right). Those are just some of the different ways in which the aspects represented by the input could combine in a unique item in the database. They comprise a movie with one actor winning an award for another movie and the second one being instead nominated for a similar award (Answer1). The second answer presents a movie where the director has received a prize instead, as well as one of the featured actors (Answer2). The third answer (Answer3) represents instead a movie where two actors have been nominated or won an award for the very same movie.

[1]developers.google.com/knowledge-graph

Contributions. To cope with the above issues, we have developed efficient algorithms that selectively construct the solution by limiting the number of isomorphic searches to be performed. Since the complete result-set may be too large to be consumed by the user and still too costly to compute, we developed a top-k solution based on a general family of ranking functions that takes into account weights on the nodes of the graph. In particular, our contributions can be summarized as follows. **(1)** We introduce and formalize the problem of answering *multi-exemplar queries* (Section II). **(2)** We propose *multi-exemplar queries* on graph-data with semantics that allow multiple combinations of non-homomorphic examples (Section II). **(3)** We describe an efficient exact method to answer exhaustively a *multi-exemplar query* on heterogeneous information graphs (Section III). **(4)** We present effective techniques for finding top-k answers given a generic relevance function defined on the nodes of the graph (Section III-D). **(5)** We illustrate the efficiency and effectiveness of our solution at scale through extensive experiments on large real world information graphs (Section IV).

II. Multi-Exemplar Queries

An *Exemplar Query* [6] is an example member of the answer set. The query engine infers the full set of answers based on the example and any semantic annotation provided by the underlying database. Here, we assume that the user presents a set of examples (also called *samples*) $\mathcal{S}$ with $|\mathcal{S}|>1$, where each one represents a partial instantiation of the features that the intended results should possess. A naïve solution for answering queries of this form is to evaluate each sample individually and return the union of the result-sets. However, this approach cannot retrieve answers that match (at the same time) more than one of the input query samples.

Therefore, we are in need of more expressive semantics: by providing several different samples, the user tries to describe results that match all their characteristics at once. We then obtain the following definition, when considering a set of (disjoint) user samples $\mathcal{S}$ in a database $\mathcal{D}$:

Definition 1 (Multi-Exemplar Query: mExQ)**.** *The result of a* Multi-Exemplar query *for the set of samples $\mathcal{S}$ on a database $\mathcal{D}$, i.e., $mExQ(\mathcal{S})$, is the set $\{a \mid \forall s{\in}\mathcal{S}.a{\approx}s\}$, where a and s are elements in $\mathcal{D}$, $\mathcal{S} \subseteq \mathcal{D}$, and the symbol $\approx$ indicates a congruence relation.*

The above definition states that an answer to a multi-exemplar query is congruent to *all* the elements in the sample set $\mathcal{S}$ through a congruence relation ($\approx$). Hence, the choice of the congruence relation determines the characteristics and the nature of the answers. Note that, in the special case where $\approx$ is an equivalence relation and all the samples have the same characteristics (i.e., $\forall s_i, s_j \in \mathcal{S}.s_i \approx s_j$), the results are the same as those obtained by searching for elements similar to (any) one of the samples. Therefore, Definition 1 is a generalization of the definition of Exemplar Query [6].

On the other hand, if the congruence relation adopted is the strict equivalence relation, Definition 1 may produce an empty result set when such condition among samples does not hold. Consider the example in Figure 1: any answer strictly equivalent (e.g., homomorphic) to the first sample is not congruent to the second or the third. Consequently, the choice for the congruence relation depends on the employed data model and the intended results.

Following Definition 1, a multi-exemplar query requires that all the elements in the sample set are congruent to each result, thus enforcing the computation of all the answers at once (i.e., `AND` semantics). Therefore, answering such queries subsumes more flexible definitions, such as the `OR` semantics, or constraints on values for the answers. In this work, we aim at providing solutions for the most onerous semantics (according to Definition 1). However, we note that alternative semantics can also be captured via minimal adaptations to the proposed methods. We elaborate on this in Section III-E.

A. Multi-Exemplar Queries on Graphs

Multi-exemplar Queries can be applied to a variety of data models and congruence relations. In this study we direct our focus towards directed labeled graphs, which naturally model relational data [16], semistructured data [17], knowledge graphs [18], and many other networks [19], [20]. Formally, let $\mathcal{L}$ be a finite alphabet of vertex and edges labels. A labeled graph is a tuple $G = \langle V, E, \ell\rangle$ where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, and $\ell : V \cup E \to \mathcal{L}$ is a labeling function from each vertex in V and edge in E to $\mathcal{L}$.

The congruence relation adopted when querying graphs through examples is the graph isomorphism between the query-sample and the answer, which represents a strict bijection between both node and edge labels. In the case of knowledge graph search (refer to Figure 1), we are more interested in finding entities and concepts that have a specific relationship structure, i.e., that are connected by specific edge labels, hence the adopted congruence relation is usually *edge-preserving graph isomorphism* [10], [6]. However, in the case of multiple samples, this idea cannot be directly applied, since answers need to be congruent to query elements with different topologies. A more appropriate choice for the congruence relation requires that an answer contains structures edge-isomorphic to each sample. Intuitively, an answer is a graph that reconciles in itself all the user samples.

We first define *edge-preserving graph isomorphism* (*graph isomorphism* in what follows) and *subgraph isomorphism*. Edge-preserving refers to searching for the same *structure* as the samples, yet dropping the node name identifiers. A *graph isomorphism* between two graphs $G_1 = \langle V_1, E_1, \ell_1\rangle$ and $G_2 = \langle V_2, E_2, \ell_2\rangle$ is a bijective function $\mu : V_1 \to V_2$ such that for every $(u, v) \in E_1$, $(\mu(u), \mu(v)) \in E_2$ and $\ell_1(u, v) = \ell_2(\mu(u), \mu(v))$, and viceversa. If a graph isomorphism exists between G_1 and G_2, we say that G_1 and G_2 are *isomorphic*, and we write $G_1 \approx G_2$. A *subgraph* of $G = \langle V, E, \ell\rangle$ is a graph $G' = \langle V', E', \ell\rangle$ such that $V' \subseteq V$ and $E' \subseteq E$. With a little abuse of notation we denote subgraphs as $G' \subseteq G$. Therefore, a sample s in the sample set $\mathcal{S}$ is a subgraph of G, i.e, $s \subseteq G$. If a graph isomorphism exists between G_1 and a subgraph G_2' of G_2, we say that G_1 is *subgraph isomorphic* to G_2, and we denote it by $G_1 \sqsubseteq G_2$. Hence $G_1 \approx G_2' \subseteq G_2 \iff G_1 \sqsubseteq G_2$. We can now define a valid answer to a multi-exemplar query on the user samples $\mathcal{S}$.

Definition 2. *An* answer *to a multi-exemplar Query represented by the user samples $\mathcal{S}$ on the database $G = \langle V, E, \ell\rangle$ is a subgraph $A \subseteq G$, such that $\forall s \in \mathcal{S}, s \sqsubseteq A$.*

By comparing Definition 1 and Definition 2, we see that the latter satisfies the premises of the former: the input query is a set of exemplar graphs and the output is the set of answer graphs that are congruent by subgraph isomorphism to all of them, i.e., they contain all the query graphs as subgraphs.

B. Problem Definition

Note that Definition 2 does not constrain the subgraph size. However, with no bounds, even the entire graph may be a valid answer, which is obviously useless. On the same token, answers should not include information (in terms of nodes and edges) that is extraneous to the user request and should represent a complete concept or situation.

Therefore, it is natural that the two properties below are satisfied: first, *connectedness*, so as the subgraphs that are isomorphic to each sample should be connected in the answer graph; and second, *consistency*, so that no additional node/edge is included into the answer graph, apart from those matching the samples.

Formally, given an answer $A : \langle V_A, E_A, \ell\rangle$ on the sample set $\mathcal{S}$, the above two properties are stated as follows.

Property 1 (Connectedness)**.** *For each two nodes $n_A, \bar{n}_A{\in}V_A$ there exists an undirected path that connects n_A to $\bar{n}_A$. Also, for each sample $s_i{\in}\mathcal{S}$, $s_i{:}\langle V_i, E_i, \ell\rangle$, there*

exists $s_j \in \mathcal{S}$, $s_j{:}\langle V_j, E_j, \ell\rangle$, $s_i \neq s_j$, *with subgraph isomorphism mapping function* μ_i *and* μ_j *respectively, such that* $\exists n_i \in V_i \exists n_j \in V_j$ *for which* $n'_A = \mu_i(n_i) = \mu_j(n_j)$, *for some* $n'_A \in V_A$. *The node* n'_A *is called a* junction *node.*

Property 2 (Consistency)**.** *For each node* $n_A \in V_A$ *there exists a node sample* $n_s \in V_s, s \in \mathcal{S}, s : \langle V_s, E_s, \ell_s\rangle$ *with subgraph isomorphism mapping function* μ, *such that* $\mu(n_s) = n_A$, *and for each edge* $(n'_A, n_A) \in E_A$ *there exists an edge* $(n_s, n_s) \in E_s$ *such that* $(\mu(n'_s), \mu(n_s)) = (n'_A, n_A)$.

Note that, by this definition, all answers are consistent with the query samples, since apart from the nodes/edges matching the samples, they contain no other additional node/edge (see Figure 1). Also, since subgraph-isomorphism is a bijection, each sample is matched by a single substructure in the answer, i.e., it contains only the minimal information to satisfy the user requirements.

Combining Definition 2 with the connectedness and consistency properties, we can now define our problem.

Problem 1 (Find all mExQ answers)**.** *Given a set of samples* $\mathcal{S}$ *on the database* $G : \langle V, E, \ell\rangle$ *find all* connected *(Property 1) and* consistent *(Property 2) answers* $A \subseteq G$ *such that* $\forall s \in \mathcal{S}, s \sqsubseteq A$.

Given a single query, the number of isomorphic graphs that exists within a large knowledge base is usually extreme. Oftentimes, the user is interested only in the top-k answers, for a specific ranking function on the answers. In this regard, we propose a definition that can employ different ranking functions. We assume a weight function on each vertex $w : V \mapsto \mathbb{R}^+$. The weights can represent user preferences, value, or query relevance, and can be easily provided by contextual data, mined from query logs, or computed using ad-hoc functions [21]. The score of an answer is given by a function $\rho : \mathcal{A} \mapsto \mathbb{R}^+$, defined as an average on the node weights.

$$\rho(A) = \frac{1}{|V_A|} \sum_{v \in V_A} w(v) \tag{1}$$

For instance, assuming a weight function $w_{deg} : V \mapsto \mathbb{N}$ that assigns to each node a score equal to that node's out-degree, in Figure 1, the score of Answer 1 would be 7/7, while for Answer 2 would be 7/8.

Any function ρ similar to average may be used for ranking (such as max, or a simple sum); however, the choice in Equation 1 favors balanced answers in terms of their relative size. Nonetheless, the solutions we study in this work are efficient for the entire family of score functions that are monotonically increasing with the weights of the node scores(Section III-D). To find answers that respect the multi-exemplar query semantics and retrieve only the top-k solutions that match the user interest we define the following problem.

Problem 2 (Find top-k mExQ answers)**.** *Given a set of samples* $\mathcal{S}$ *on the database* $G{:}\langle V, E, \ell\rangle$, *a user-defined parameter* k, *a weight function* $w{:}V \mapsto \mathbb{R}^+$ *and the ranking function* ρ, *return the* k connected *(Property 1) and* consistent *(Property 2) answers* $\mathcal{A}{:}\langle A_1, ..., A_k\rangle$, *such that* $\forall A_i \in \mathcal{A}$, *all the following hold: (i)* $A_i \subseteq G$, *(ii)* $\forall s \in \mathcal{S}. s \sqsubseteq A_i$, *and (iii) given any other answer* $A' \notin \mathcal{A} \wedge \forall s \in \mathcal{S}. s \sqsubseteq A' \rightarrow \rho(A') \leq \rho(A_i)$.

Algorithm 1 `mQ-Naive`

```
1: function PARTIAL(G, S)
2:     return G : {G}                          ▷ Selectively pruned based on S
3: function SEARCH(G, S)
4:     A ← ∅
5:     for each G ∈ G do
6:         for each s_i ∈ S do                 ▷ Find isomorphic subgraphs
7:             Ã_i ← {A ⊆ G | s_i ≈ A}
8:         C ← arg min_{Ã_i ∈ ⟨Ã_1,...,Ã_|S|⟩} |Ã_i|
9:         while C ≠ ∅ do
10:            c ← REMOVEONE(C)
11:            if ∀s ∈ S.s ⊑ c then            ▷ Check sample in c
12:                A ← A ∪ {c}
13:            else C ← C ∪ CONNECT(c, S, ⟨Ã_1, ..., Ã_|S|⟩)
14:    return A
```

Algorithm 2 CONNECT$^+$

Input: Candidate $c : \langle V_c, E_c, \ell_c\rangle$; HashTable $\mathbb{H}$
Output: Expanded candidates C^+

```
1: C+ ← ∅
   ▷ Find candidate nodes contained in some answer
2: for each n ∈ V_c do
3:     for each Ã_i ∈ H(n) s.t. s_i ⋢ c do
4:         C+ ← C+ ∪ MERGE(c, Ã_i)
5: return C+
```

III. Proposed Approach

We start with input a set of disconnected query-samples. Note that there exist already a number of methods to obtain graphs representations of the user requirements [22], [23]. We design Multi-Exemplar Queries Answering as a two step approach, using the PARTIAL and SEARCH functions. The first step takes as input the graph and detects a set of candidate subgraphs, or regions $\mathcal{G}$, that most probably contain multi-exemplar answers. The second step searches for answers in such candidate regions. In what follows, we describe algorithms for these two steps.

A. The Baseline Algorithm

As a baseline approach for Multi-Exemplar Query Answering (Problem 1), we extend the Exemplar Query approach [6] and apply some additional optimizations. We refer to this algorithm as `mQ-Naive` (shown in Algorithm 1). Here, the PARTIAL step returns only one candidate subgraph that corresponds to the whole graph, eventually pruned of edges that do not appear in the input.

The SEARCH function, instead, first finds the partial matches (line 7) and then joins them (line 13). In particular, it finds the graphs isomorphic to each sample individually (line 7 can involve any graph isomorphism algorithm and its optimizations [24], [25]), obtaining $|S|$ sets, which are the candidate partial answers $\langle \tilde{A}_1, ..., \tilde{A}_{|\mathcal{S}|}\rangle$. Subsequently, each individual graph is combined with the others (lines 13) to fulfill the *Connectedness Property* (Property 1, Section II), keeping only those that can be

merged into a complete answer (lines 11-12). To speed up the computation, we avoid the Cartesian product $\tilde{A}_1 \times \tilde{A}_2 ... \times \tilde{A}_{|\mathcal{S}|}$ of all possible combinations of individual sample answers for verifying when Property 1 holds. In our algorithm, instead, we progressively expand the smaller set of answers from one single sample (line 8) and, by enforcing Property 1, we merge answers $\langle \tilde{A}_1, ..., \tilde{A}_{|\overline{\mathcal{S}}|} \rangle$ from other samples $\overline{\mathcal{S}} = \mathcal{S} \setminus \{s_i\}$ until no other merge is possible.

Baseline Optimizations. We now describe some optimizations we apply to our baseline solution. First, we avoid checking the entire graph in the PARTIAL function. Instead, we selectively load only the portion that most probably contains answers to the multi-exemplar query, as follows. The edge-labels appearing in the samples are sorted according to their frequency in the graph. The less-frequent label is first considered, and the corresponding edges in the graph are loaded. We then select neighbor edges in the samples and in the graph such that the edges selected in the graph have the same label as those in the samples. The procedure stops when all the edges in the samples have been considered. Only the connected components that contain all the edge-labels are returned, and we avoid loading portions of the graph that will not match any sample. To further improve this procedure we store in advance the set $\{x \in V | (x,y) \in E \wedge \ell(x,y) = l\}$ of nodes that are source of edges with any label l along with the number of occurrences of l in the graph.

We also introduce a second optimization in the CONNECT function (line 13, Algorithm 1). The role of CONNECT is to expand a candidate answer c with all the possible answers from individual samples that share a node with c. A straight-forward implementation where we check all partial graphs will lead to a very high computational cost. Algorithm 2 efficiently solves this problem, employing a hash-map $\mathbb{H}$ on the nodes of the answers to each sample. The hash-map is computed first: for all nodes in the graphs that are isomorphic to (any of) the samples, it maps to the set of graphs that contain this node. That is, for a node n, $\mathbb{H}(n) = \langle \tilde{A}_1^{(n)}, ..., \tilde{A}_{|\mathcal{S}|}^{(n)} \rangle$, such that $\tilde{A}_i^{(n)} = \{A : \langle V_A, E_A, \ell_A \rangle | A \subseteq G, A \approx s_i, n \in V_A\}$. The hash-map stores only those nodes that appear in at least two graphs for two different samples. The algorithm then retrieves for each node in the candidate c only those answers that can be connected to it, and for which the corresponding sample is not already subgraph isomorphic to c (line 3). This is achieved by annotating c with the list of matching samples.

Complexity Analysis. We have $n = |\mathcal{S}|$ samples, and for each sample $s_i \in \mathcal{S}$, $|\tilde{A}_i|$ is the number of isomorphic answers to s_i. Then, the computational cost of the algorithm is at least the cost of solving n times subgraph isomorphism, which is NP-complete [26], and then the worst case performance of all the calls to CONNECT sums up to $\mathcal{O}(\prod_i^n |\tilde{A}_i|)$. Yet, with our optimization we reduce it to $\mathcal{O}(|\tilde{A}_{min}| \times |\mathcal{S}| \times \bigcup_i^n V_{\tilde{A}i})$, where $|\tilde{A}_{min}|$ is the smallest set of partial answers, and $V_{\tilde{A}i}$ the union of all the nodes among all the partial answers for s_i, which is in turn bounded by the set V of all the nodes in the graph..

B. Finding Answers Efficiently

`mQ-Naive` has two main bottlenecks: (1) the computation of all the individual sample answers, and (2) the need to compute and store all the possible partial answers that are built during the incremental expansion of candidates.

We propose here a more efficient (exact) algorithm (Algorithm 3), called `mQ-Fast`, which first selects subgraphs matching one single sample, and then selectively expands these subgraphs in search for complete answers. This approach can reduce the number of isomorphism evaluations, and the number of graphs kept in memory at each step. In particular, we show here the modified implementation of PARTIAL function for the retrieval of candidate subgraphs.

The expansion step starts from a sample (with the minimum number of expected appearances in the graph, line 2) and retrieves the nodes of its matching subgraphs (line 3), these are partial answers. To select the initial sample, we estimate the number of matching subgraphs for a graph by exploiting edge statistics as explained later.

From the nodes of the partial answers computed, we start a constrained expansion (EXPAND - line 11-25). The expansion includes neighboring nodes, in a breadth-first fashion, while retaining only those that potentially belong to one of the partial answers for the other samples, until no other neighbor node is added (line 5-8). This exploration exploits a compact representation of the edge-labels in the neighborhood of each node at some distance d (usually at most 3 [6]), called *node-vectors*. The expansion procedure compares the node-vectors of the samples to the node-vectors in the current candidate. Thus, we can exclude non-matching nodes by comparing vectors instead of graph structures. The candidate subgraphs $\mathcal{G}$ obtained at the end of this procedure are then passed to the SEARCH procedure in Algorithm 1 (which we described earlier for `mQ-Naive`).

Node-vectors. The node-vectors representations are computed as follows. We assume the labels to be ordered, i.e., $l_1, ..., l_{|\mathcal{L}|}$. Given a node n, and a maximum distance value d from n, we compute vector $\mathbf{v}^{(n)} = [v_1^n, ..., v_M^n]$, where $M = d|\mathcal{L}|$ represents the number of entries in the vector (i.e., one for each label at each distance). An entry in the vector is set to $v_i^n = 1$ iff there is at least one edge labeled l_t, where $t = (i \mod |\mathcal{L}|) + 1$ at distance $\lfloor i/d \rfloor$ (see the upper part of Figure 2, where 0s are replaced by "$-$" for readability). We note experimentally that the number of edge-labels in real large graphs with more than 10^7 nodes is usually below 10^5, and, given the high connectivity of the graph, considering distances above three hops provides limited gain. Consequently, the size of these vectors is also limited. Moreover, these vectors are usually sparse, allowing for a considerable space reduction.

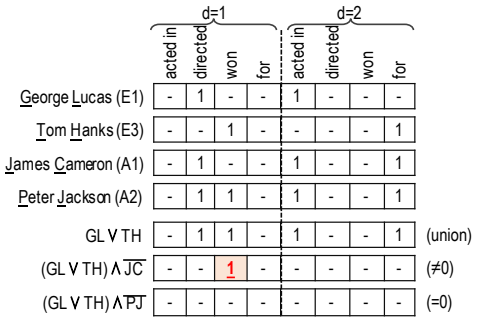

Fig. 2. Node Binary Vector Matching.

For instance, in a real graph with $4k$ labels [27] each node uses on average up to $10d$ bits.

The vectorial representation provides an effective way to compare a node from a candidate answer with a node from a sample. A graph node is a candidate matching for a sample node if the two vectorial representations are *compatible*. The vectorial representations are compatible if the candidate matching node has a 1 in the same positions as the sample node vector. This is assessed through fast bitwise AND operations between the negation of the node vector and the sample node vector. More formally, given the vectorial representation $\mathbf{v}^{(n)}$, we denote as $\bar{\mathbf{v}}^{(n)}$ the bitwise-negated version of $\mathbf{v}^{(n)}$, i.e., $\bar{v}_i^n = 1 - v_i^n$. We also write $\mathbf{v}^{(n_1,n_2)} = \mathbf{v}^{(n_1)} \vee \mathbf{v}^{(n_2)}$ as the union vector between n_1 and n_2, where $\vee$ is the bitwise OR (similarly $\wedge$ indicates the bitwise AND). Hence, a node n is a candidate node matching the sample node n_1 if $\mathbf{v}^{(n_1)} \wedge \bar{\mathbf{v}}^{(n)} = \mathbf{0}$ (i.e., the zero vector). Then, using the distributive property of logical conjunction over disjunction we have an effective method to assess if a node might connect two or more samples. In particular, if a node n is a candidate node shared between two samples nodes n_1, n_2 from sample s_1 and s_2, respectively, the following equation holds.

$$(\mathbf{v}^{(n_1)} \vee \mathbf{v}^{(n_2)}) \wedge \bar{\mathbf{v}}^{(n)} = \mathbf{0} \tag{2}$$

Hence, we check if a node can be a junction node without any false negatives (i.e., we never discard nodes that are part of an answer).

A simplified example is shown in Figure 2, with reference to Figure 1. We see a vectorial representation, $d = 2$, of the nodes *George Lucas* (GL) from *Example*1 (E1), *Tom Hanks* (TH) from *Example*3 (E3), *James Cameron* (JC) from *Answer*1 (A1) and *Peter Jackson* (PJ) from *Answer*2 (A2). The union vector of GL and TH is GL∨TH. We then see that node JC is not a joint for the two, because it cannot match an edge labeled *won* for $d = 1$, so the result of the bitwise operation $(\mathsf{GL} \vee \mathsf{TH}) \wedge \overline{\mathsf{JC}}$ is not the zero vector $\mathbf{0}$. On the other hand, the node vector for PJ can match all the necessary edge labels, thus $(\mathsf{GL} \vee \mathsf{TH}) \wedge \overline{\mathsf{PJ}} = \mathbf{0}$, and PJ is identified as a junction node.

We now formally prove that any matching node possesses the above property.

Theorem 1. *Let $A : \langle V_A, E_A, \ell_A \rangle$ be a multiple exemplar answer for samples $s_1 : \langle V_1, E_1, \ell_1 \rangle$, $s_2 : \langle V_2, E_2, \ell_2 \rangle$ in a database $G : \langle V, E, \ell \rangle$. If $n \in V_A$ is a matched node shared among s_1 and s_2, then exist two nodes $n_1 \in V_1, n_2 \in V_2$ such that $\mathbf{v}^{(n_1,n_2)} \wedge \bar{\mathbf{v}}^{(n)} = \mathbf{0}$.*

Proof: (sketch) If n is the matching node between n_1 and n_2 by Property 1 it belongs to the subgraph isomorphism relations of s_1 and of s_2 to A. Therefore, both the structures surrounding n_1 and n_2 are included in the neighbors of n, i.e., we can follow any undirected-path starting from either n_1 or n_2 in the respective samples, and we will find, at any hop distance, a path with the same labels starting from n in A. Consequently, given the binary vectors $\mathbf{v}^{(n_1)}$ and $\mathbf{v}^{(n_2)}$, it holds that $\mathbf{v}^{(n_1)} \wedge \bar{\mathbf{v}}^{(n)} = \mathbf{0}$ and $\mathbf{v}^{(n_2)} \wedge \bar{\mathbf{v}}^{(n)} = \mathbf{0}$. Hence, it follows that it must hold true $((\mathbf{v}^{(n_1)} \vee \mathbf{v}^{(n_2)}) \wedge \mathbf{v}^{(n)}) = (\mathbf{v}^{(n_1)} \vee \mathbf{v}^{(n_2)}) \implies (\mathbf{v}^{(n_1)} \vee \mathbf{v}^{(n_2)}) \wedge \bar{\mathbf{v}}^{(n)} = \mathbf{0}$ ■

Algorithm 3 mQ-Fast

```
1: function Partial(G, S)
2:     s* ← Select(S)                    ▷ Choose the best starting sample
3:     C ← {A ⊆ G | s* ≈ A}
4:     G ← ∅
5:     while C ≠ ∅ do
6:         c ← RemoveOne(C)
7:         if S \ {S_c} ≠ ∅ then
8:             C ← C ∪ Expand(c, S \ S_c, G)
9:         else G ← G ∪ {c}
10:    return G
11: function Expand(c, S̄, G)            ▷ Add matching nodes to c
12:    toVis ← Vis ← V_c ← Maps(V_c, S̄)
13:    L_S ← {ℓ(e_s) | ∀s ∈ S̄ ∀e_s ∈ E_s}
14:    while toVis ≠ ∅ do                ▷ Pruning BFS
15:        n_c ← RemoveOne(toVis)
16:        V_t ← {x | (n_c, x) ∈ E, ℓ(n_c, x) ∈ L_S, x ∉ Vis}
17:        V_c ← V_c ∪ Maps(V_t, S̄)
18:        Vis ← Vis ∪ V_t; toVis ← toVis ∪ V_t
19:    V_t ← {v | v ∈ ⋃_{s∈S̄} V_s, ∃n ∈ V_c s.t. v^(v) ∧ v̄^(n) = 0}
20:    if ⋃_{s∈S̄} V_s \ V_t = ∅ then
21:        return {G[V_c]}               ▷ Subgraph of G induced by V_c
22:    return ∅
23: function Maps(V_t, S)                ▷ Filter non matching nodes in V
24:    V_c ← ∅
25:    for each v ∈ V_t do
26:        for each n_s ∈ ⋃_{s∈S} V_s s.t. v^(n_s) ∧ v̄^(n_c) = 0 do
27:            V_c ← V_c ∪ {v}
28:    return V_c
```

Cardinality Estimation. We now describe our solution for cardinality estimation of isomorphic subgraphs (Algorithm 3, Line 2), to return the sample with the minimum number of expected matches. Existing models for selectivity of graph queries and cardinality estimation of their results [28], [29] are designed to capture complex interdependencies between labels and nodes and to estimate the size of the results of specific graph queries. Also, these approaches heavily exploit attributes in nodes and edges. Hence they do not easily adapt to our case, where edge-label connectivity is the only information that we can exploit.

To compute our estimation we first decompose a graph into a set of star structures, as it is also done for graph

query answering [30], i.e., trees with a single node and n children at depth 1. Computing cardinality-upperbounds for those small trees is easy, as we can exploit the frequency of co-occurrence of label-pairs. Given the maximum match cardinality for each star, we approximate the number of matching based on the combination of those upperbounds.

We maintain two cardinality indexes to quickly estimate the selectivity of edge labels and their co-occurrence: $\mathbb{I}_{pair}$ and $\mathbb{I}_{star}$. The first one, $\mathbb{I}_{pair}$, maintains the number of occurrences of patterns composed by just two edges. Thus, for each pairs of labels $l_1, l_2 \in \mathcal{L}$ the index stores $\mathbb{I}_{pair}(l_1,l_2)=\{G' \subseteq G | G' : \langle V', E', \ell\rangle,\ E' = \{(v_1,v_2), (v_2,v_3)\}$ s.t. $\ell(v_1,v_2) = l_1,\ \ell(v_2,v_3) = l_2\}|$. The edge labels are hashed to speedup retrieval, and the entire data structure can fit in memory since its size is limited by $\mathcal{O}(|\mathcal{L}|^2)$. Note that the number of pairs is usually much smaller, as not all combinations exist in the graph. Also, in real graphs $|\mathcal{L}|$ is less than 10^5.

The second index, denoted as $\mathbb{I}_{star}$, stores the number of occurrences of a star subgraph containing a label $l \in \mathcal{L}$, having a predefined size $c > 0$. Therefore the index $\mathbb{I}_{star}(l,c) = |\{G' \subseteq G | G' : \langle V', E', \ell\rangle$ is a star $\wedge |E'| = c \wedge \exists (v_1,v_2) \in E'$ s.t. $\ell(v_1,v_2) = l\}|$. The size of this index is bounded by the number of labels $|\mathcal{L}|$ and the maximum c, which in our case is determined by a parameter C_{max}. Hence, the index size is $\mathcal{O}(C_{max}|\mathcal{L}|)$, but for all practical cases $\mathcal{O}(|\mathcal{L}|)$, since usually $C_{max} \ll |\mathcal{L}|^2$.

The cardinality estimation works as follows. If the sample is just an edge, then the frequency of the label is the correct estimation. If the sample is a 2-edges path, then the index $\mathbb{I}_{pair}(l_1,l_2)$ stores the correct frequency of such graphs. To estimate an upperbound for the cardinality of a star-shaped sample $G^* : \langle V^*, E^*, \ell^*\rangle$ we first compute the maximum number of stars that can exists with $|E^*|$ edges and at least one of them with label l, which is computed as follows $Stars(l) = \sum_{c=|E^*|}^{C_{max}} \mathbb{I}_{star}(l,c) * \binom{c}{|E^*|}$

The summation takes into account that $\mathbb{I}_{star}$ contains values for different numbers of edges ($c \in [1, C_{max}]$). For $c = |E^*|$ then $\mathbb{I}_{star}(l,c)$ is the number of stars with exactly $|E^*|$ edges. Then we take into account stars that are formed by selecting a subset from a star with any $c > |E^*|$. In this case, we consider the number of subsamples of size $|E^*|$ out of c elements.

By selecting a label $l_1 = \arg\min_{l \in \mathcal{L}_{G^*}} Stars(l)$, we know that $Stars(l_1)$ is an upper-bound estimation for the number of subgraphs isomorphic to G^*. To obtain a much tighter upper-bound, although approximate this time, we exploit pair-label frequencies once more. We select a second label to be $l_2 = \max_{l \in \mathcal{L}_{G^*}} \mathbb{I}_{pair}(l_1,l)$, i.e., the label that more often appears paired with the previously selected. We estimate the selectivity of G^* as the number $Stars(l_1)$ scaled by the conditional probability of finding l_2 given l_1. This is justified by the fact that not all the stars that have the correct size and contain the label l_1 also contain l_2, while both are required by G^*. The final (estimated) selectivity of G^* is then

$$Stars(l_1) * \frac{\mathbb{I}_{pair}(l_1,l_2)}{\sum_{l \in \mathcal{L}} \mathbb{I}_{pair}(l_1,l)}. \quad (3)$$

For more complex structures, we estimate the selectivity of the graph as the lowest selectivity among its stars. We experimentally demonstrate the accuracy of this estimation (Section IV).

Complexity Analysis. The `mQ-Fast` algorithm does not discard any correct answers (Theorem 1). In terms of time complexity, the most demanding tasks are `Maps`, `Expand`, and subgraph isomorphism. `Maps` compares the node-vectors in V with each sample node-vector, which takes $\mathcal{O}(d|\mathcal{L}| \sum_{s \in \mathcal{S} \setminus \bar{\mathcal{S}}} |V_s|)$. The `Expand` procedure instead performs a traversal of the graph for nodes matching a single sample. This procedure is then repeated at every cycle. In the worst case lines 5-10 in `Partial` scan the entire graph, leading to a complexity of $\mathcal{O}(d|\mathcal{L}| \sum_{s \in \mathcal{S} \setminus \bar{\mathcal{S}}} |V_s| \times |V|^2 |\mathbf{C}|)$.

C. Avoiding Redundant Computations

We observe that the `mQ-Fast` algorithm performs several expensive operations when generating candidate answers, at the very beginning of the `Partial` function.

We now present the `mQ-Fast`$^+$ algorithm (Algorithm 4), which introduces further optimizations, while still producing all answers. First, we observe that thanks to the expansion process, retrieving all possible answers for a sample is not necessary, as long as one single node on the candidate sample is considered. In the `mQ-Fast`$^+$ algorithm, the `Partial` function finds the (candidate) node matchings between sample nodes of a selected sample (line 3) and graph nodes using the node-vectors (line 4-5). In particular, it chooses the node of the selected sample with the minimum number of matches (line 6-7) and uses each on of those matching as seeds for expansion. This allows avoiding performing the expensive isomorphic search at the beginning. Second, we note that the candidate returned in the `Expand` function may be generated multiple times, if some multi-exemplar answers overlap. This occurs when one candidate includes the answers of another candidate that has already been processed. To prevent this, we add an extra condition, which removes from the list a candidate that overlaps with another one (line 11). The `Expand` function checks if we have found all the matchings for all the samples as in `mQ-Fast` (line 23, Algorithm 3).

Complexity Analysis. This optimization does not require any extra indices, but all the optimizations proposed for `mQ-Naive` and `mQ-Fast` can be used in `mQ-Fast`$^+$, as well. The complexity remains the same as before, as lines 4 to 7 iterate over each node in the graph, with $\mathcal{O}(d|\mathcal{L}| \times |V|)$ operations.

[2]In our experiments, $C_{max} = 10$ takes into account the average node-degree and the structure of the expected queries.

Algorithm 4 mQ-Fast$^+$

```
1: function PARTIAL(G, S)
2:     G ← ∅
3:     s* ← SELECT(S)
4:     for each n ∈ V_{s*} do
5:         M(n) ← {v ∈ V | v^(v) ∧ v̄^(n) = 0}
6:     n* ← arg min_{n∈V_s} |M(n)|        ▷ Node with min-matchings
7:     C ← M(n*)
8:     while C ≠ ∅ do
9:         c ← REMOVEONE(C)
10:        c+ ← EXPAND(c, S, G)
11:        C ← C \ {c1 ∈ C | c1 ⊑ c+}      ▷ Remove redundancy
12:        G ← G ∪ c+
13:    return G
```

D. Finding Top-k Answers

In this section, we consider the problem of returning only the k best answers to a multi-exemplar query, given a generic scoring function on the graph nodes (Problem 2). This is fundamental in the contex of large graphs, where too many results would overload the user. Note that the algorithm proposed in this section is exact, i.e., the returned answers have the k highest scores.

In order to compute the score of an answer, Section II describes a reasonable instance for the score function that computes the average between the weights of the elements to prevent the side effect of skewing the result-set in favor of larger results. Yet, we note that any other analogous function, which can be bounded by monotonically higher ranking scores for answers containing higher scoring nodes, can be used. Here, we introduce an early termination method, based on the upper bound for the ranking function in Equation 1 that can be computed in any given part of the graph. Thus, we avoid SEARCH computing isomorphic graphs in all the areas selected by PARTIAL where answers are bound to a ranking score that is too low. The procedure can stop searching as soon as the kth lower scoring answer A_k found has a score $\rho(A_k)$ higher than any upperbound $\overline{\rho}$ for the remaining portions of the database. Therefore, given a portion of the graph G, we aim at pairing each sample node with a graph node, such that the resulting scoring function is maximized. The optimization version of this problem can be seen as weighted formulation of the hitting set problem [31], which makes a tight estimation impractical. Instead, we propose the upperbound $\overline{\rho}$ computed according to the following theorem.

Theorem 2. *Given the set of graph samples $\mathcal{S}$, and answers, A_1 and A_2, $\forall s \in \mathcal{S}. s \sqsubseteq A_i$, via the isomorphism function $\mu^s_{A_i}$, the node weight function w, and the ranking ρ (Equation 1). It holds:*

$$\overline{\rho}(A_2) = \frac{\sum_s^{\mathcal{S}} \sum_v^{V_s} w(\mu^s_{A_2}(v))}{max_{s\in\mathcal{S}}|V_s|} < \rho(A_1) \rightarrow \rho(A_2) < \rho(A_1) \quad (4)$$

Proof: (sketch) (1) Some $v \in V_{A_2}$ are junction nodes and match more than one sample, we have that $\sum_s^{\mathcal{S}} \sum_v^{V_s} w(\mu^s_{A_2}(v)) > \sum_v^{V_{A_2}} w(v)$. (2) Then, given that an answer contains all samples, we know that $\forall s \in \mathcal{S}. |V_{A_2}| \geq |V_s|$. It follows that $\rho(A_1) \geq \overline{\rho}(A_2) \geq \rho(A_2)$ ■

Hence, given a portion of the graph $G'$for which to estimate the upperbound of the scoring function, for each node v_s in each sample $s \in \mathcal{S}$, we select the candidate matching $n \in V_{G'}$ with the highest weight $w(n)$. Note that, given Theorem 1, we can use the vector representation of each sample and each candidate node to recognize which node could be used for the mapping. We compute the score for a node v_s as $max_{\mathbf{v}^{(v_s)} \wedge \bar{\mathbf{v}}^{(n)} = \mathbf{0}} w(n)$

We change the SEARCH procedure described earlier to compute in advance such upper bound, and search first within the region that has the largest one. The optimization proposed here is implemented in a modified SEARCH function. Note that, the difference between the exhaustive solutions mQ-Fast and mQ-Fast$^+$ is the implementation of PARTIAL. Hence, the SEARCH, is applicable to both of them, obtaining in this way mQ-Fast-topK and mQ-Fast-topK$^+$.

Weight functions. Even though the study of the best ranking is out of the scope of this paper, to showcase the flexibility of our approach, we implemented a set of traditional measures adopted in top-k algorithms for different use-cases. In particular, we consider (1) a *degree-based* ranking function, which uses the node degree as the weight for each node so that the popular nodes are those that are the highest probability to be found [32]; (2) *structural similarity* computing the jaccard similarty of the sets of labels at distance d from each node, or alternatively computing the maximum cosine similarity between the vectorial representations of the neighborhood of each node [21]; finally (3) a *random-walk* similarity measure provides higher score to those nodes where most probably a random surfer will end up when starting from the nodes in the query. In other cases we readily exploit precomputed weights, e.g., from query-logs or other statistics [33].

E. Alternative semantics

We previously focused on finding all, or top-k results that are congruent to *all* the samples at the same time. This section presents immediate adaptations to the proposed techniques to accept alternative semantics, yet preserving Connectedness (Property 1) and Consistency (Property 2) of the answers. Although an exhaustive study of alternative semantics is out of the scope of the current work, we describe two extensions that fit a vast number of use cases: optional samples, and fixed node labels.

Optional samples. Optional samples refer to the situation in which the user can specify whether the samples should be part of the multi-exemplar answers, or not. This case reflects the OR and the OPTIONAL clauses in SPARQL queries [3]. The OR clause requires that at least one of the samples in the clause is in the answer. The OPTIONAL clause additionally allows the case that none of the samples in the clause are in the answer. Moreover, any combination of OR, OPTIONAL and AND (our proposed semantics) clauses is also taken into account.

To adapt our current framework to these more flexible semantics, we need to consider the logic formula $\varphi_{\mathcal{S}}$ expressed as `AND` and `OR` clauses over the samples $\mathcal{S}$. For instance, if the user requires sample s_1 and one among samples s_2 and s_3, the formula over $\mathcal{S}=\{s_1, s_2, s_3\}$ is $\varphi_{\mathcal{S}}:=s_1 \wedge (s_2 \vee s_3)$. `OPTIONAL` clause are not considered in the formula since they are not required for consistency. Then, `mQ-Naive` (Algorithm 1) instead of considering valid candidates (c, Line 11) that contain all samples, should check whether they satisfy the formula $\varphi_{\mathcal{S}}$. We change `mQ-Fast` and `mQ-Fast`$^+$ preventing an early pruning of potentially good candidates (Algorithm 3, Line 20). A simple adaptation removes the pruning condition (Line 20), while a more elaborate solution first converts the formula into Conjunctive Normal Form (CNF), and then checks whether the sample is at least in one `AND` clause.

Fixed node values. The fixed node semantics allow the specification of fixed nodes, or edge values in the samples. For instance, the user might be interested in movies, where the director is always George Lucas. Such constraints can be easily included in the current solution by means of additional conditions in the graph isomorphism. More specifically, the number of candidate answers for each sample (refer to Algorithm 1, Line 7), is conditioned to the values expressed by the user, exclusively. This additional condition can substantially speed up the computation of answers for multi-exemplar queries.

IV. Experimental Evaluation

Since the solutions presented in this work are exact, we focus on the efficiency of the proposed optimizations, both for the computation of the complete result set (Problem 1) as well as for the top-k answers (Problem 2). We also report on the quality of our selectivity estimation, compare the efficiency of our optimization to a solution for partial topology matching, and demonstrate the expressiveness of the paradigm with results over a real knowledge graph.

Datasets: We tested our algorithm on two of the largest existing knowledge graphs: Freebase [27] and Yago [34]. We downloaded both graphs in their latest version and removed the unnecessary metadata (e.g., users information and multilingual names). We obtained for **Yago3** (`YG` from now on) 2.9M nodes (comprising entities and taxonomy) and 16.7M edges with 38 edge labels. **Freebase** (`FB` in the following) instead is much larger, it contains a graph of 76M nodes and 314M edges, with about 4.5K distinct edge labels. We also compared to PANDA [14], on one of the datasets used in their evaluation, **Cit-HepPh** [35]: a citation network of papers, with a total of $\sim 30K$ papers (nodes) and $\sim 35K$ citation links (edges). Each paper is a node with the publication month and year as a label. The original dataset had 122 node labels; we assigned to each edge a label obtained by concatenating the two labels of the edge vertices. In this way, we obtained 8114 distinct edge labels without changing the structure of the graph.

Queries: Since no existing real-world benchmark is available for the problem of multiple-example graph queries, we collected query samples via a user study asking 20 users to create multiple-example queries on different topics, such as movies, countries, politics, and so on. To this end, we instructed the users on searching a prototype search engine and looking for entities and connections among them. The users were partially volunteers and partially hired through a crowd-sourcing platform[3]. The queries obtained as described represent the first real dataset for Multiple Exemplar Query, which we now make available[4].

Based on the structure and size of the obtained real queries, we generated a workload of single connected subgraphs of size 1 to 6 edges, based on both the `YG` and `FB` knowledge graphs. Following previous works [6], [14], this was done via random-walk sampling. We then combined these subgraphs in sets of different multi-exemplar queries. We generated multi-exemplar queries of each sample size between 2 and five samples, starting from 5-samples queries and repeatedly subsetting the samples to obtain the smaller sets, resulting in 160 queries for `FB` and 120 for `YG`. In our experiments, we report results based on a subset of 100 queries (25 for each one of the four different sample sizes) that all algorithms can fully process in memory. The size of this query-workload is among the largest in this field [6], [14], [10], and we make it available online[4]. We did the same for the *Cit-HepPh* dataset, for which we additionally built queries with 6 and 8 subgraphs.

Experimental Setup: All algorithms presented in this paper were implemented in Java 1.8, on an Intel Xeon $E52440$ (12 Cores 2.40GHz, 188Gb RAM) server running Linux $v3.13.0$. Regarding PANDA [14], we obtained the code from the authors, and with their feedback, we applied the changes described in their paper to allow also for answers similar to our semantics. Similar to other approaches, the knowledge graphs and all relevant indexes are memory resident [6].

A. Results

Selectivity estimation: First, we evaluate our selectivity estimation method (described in Section III-B) concerning the real number of subgraph isomorphic structures, since the selectivity estimation takes negligible time (< 10ms), but is an important component in our optimizations. To this end, we compare the cardinalities of all the generated samples to the estimates produced by our method. Both estimated and actual cardinalities are sorted by the number of answers. The closer the two rankings are, the more likely a pairwise comparison between samples provides the correct minimum. Therefore, we measure the Spearman's rank correlation [36] between the two ranked lists: a high correlation value would mean that our method obtains a ranking similar to the real one. The result of the

[3] www.crowdflower.com

[4] disi.unitn.eu/~lissandrini/files/mexq-queries.zip

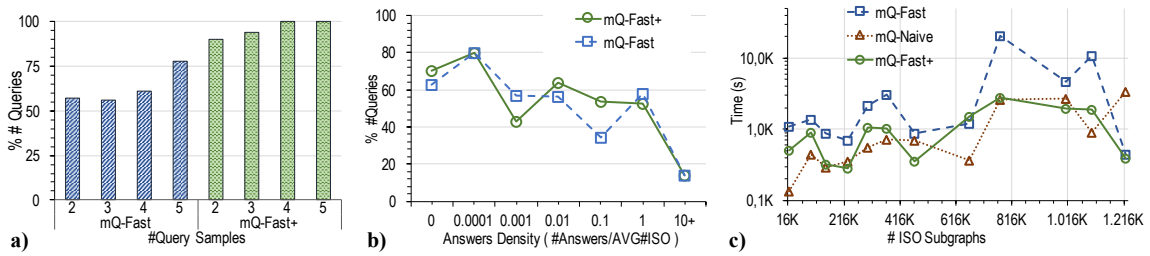

Fig. 3. Left: portion of queries where `mQ-Fast` and `mQ-Fast`$^+$ compute less isomorphism than `mQ-Naive` as a function of the number of fragments a); and of the ratio of final answer over the number of isomorphic subgraphs in the graph b). Right: c) Comparison of Running time w.r.t. number of isomorphic graphs existing in the graph.

experiments (omitted for brevity) shows a Spearman rank correlation of 0.81 with p-value $< 10^{-50}$. This means that in most cases our estimate can identify the best sample to select for our algorithms.

Evaluation of Complete Search: Considering that `mQ-Naive` retrieves all the subgraphs isomorphic to all samples, we test how many of those `mQ-Fast` and `mQ-Fast`$^+$ computes. In Figure 3.a, we show the percentages of queries (on `FB`) in which our optimized algorithms compute fewer isomorphisms than `mQ-Naive` as a function of the number of samples. The results show that `mQ-Fast` computes the same (or more) number of isomorphisms than `mQ-Naive` in 43% of cases, while `mQ-Fast`$^+$ is more efficient in $> 90\%$ of the cases. Those are cases in which some structures are shared by many answers. Hence they appear in many different candidate regions of $\mathcal{G}$ when computed by PARTIAL, in these particular cases `mQ-Fast` is wasting some computations. Note that, despite more subgraphs are computed, they are generated in different iterations, so that, at any time, only a portion of those is in memory. `mQ-Fast`$^+$, on the other hand, will never waste computation since the regions $\mathcal{G}$ identified by the optimized PARTIAL (Algorithm 4) never overlap. Hence, it computes at most the same number of subgraphs computed by `mQ-Naive` but never more proving that our optimizations reduce the memory requirements of the algorithms, leading to better scalability than `mQ-Naive`.

In Figure 3.b, we present the percentages of queries (on `FB`) in which the two algorithms, `mQ-Fast` and `mQ-Fast`$^+$ compute faster than `mQ-Naive` as a function of the ratio between the number of all final answer (without top-k) and the average number of isomorphic graph per sample. In the presence of few multi-exemplar answers, even if there are many candidate fragments, the optimizations are faster in more than 50% of the cases, and can still be faster in less favorable situations. Indeed, when there are more multi-exemplar answers than fragments (and the ratio is >1), it means that few fragments combine in many different ways, so it is better to compute the few fragments and then compute their combinations. This is also shown in Figure 3.c, where we show the running time on `FB` as a function of the sum of isomorphic subgraphs present in the knowledge-base (here, points summarize intervals of approximately $60K$). Hence, our choice falls on `mQ-Fast`$^+$ for large knowledge graphs with rich information, where `mQ-Naive` would not cope with the number of candidates to handle and `mQ-Fast` has a higher risk to waste computations. Also, this suggests that we could apply of strong-simulation [37] in place of isomorphism as the congruence relation, allowing for many solutions to be merged in one single answer [6], [14].

Evaluation for Top-K Search: We first tested the performance obtained to compute the top-3 answers in the `FB` dataset with `mQ-Fast` and `mQ-Fast`$^+$, and compared that to the baseline `mQ-Naive`. We report values only for the *structural similarity* weight function since the behavior of the other functions is comparable. Figures 4.a and 4.b show the median number of isomorphisms and the median and average running-time as a function of the number of samples, respectively. Note that the query time accounts for the complete process of retrieval and ranking of the answers. While query time is biased towards the specific implementation, the number of isomorphisms is not.

The median number of isomorphisms (Figure 4.a) shows that both optimizations reduce the number of computations most of the times. As seen before, on average (not shown in the figure) `mQ-Fast` computes many more isomorphisms than `mQ-Naive`. This is also reflected in the average running time (Figure 4.b). The different behavior of `mQ-Fast` on average and median reflects once again the larger sensitivity of the method to the graph topology. As a matter of fact `mQ-Fast`$^+$, performs up to two times better than `mQ-Naive` regarding the number of isomorphisms and time. Therefore, for these queries `mQ-Fast`$^+$ is the only choice. Finally, we report that experiments on `YG` obtained similar results (Figure 4.c). Yet, on `YG`, the difference between `mQ-Fast` and `mQ-Fast`$^+$ is rather negligible, while the gain of `mQ-Fast`$^+$ is much larger on `FB`. This shows that (1) the algorithms keep similar performances on the larger and the small graph, but also (2) that it is not just the size of the graph but also the number of isomorphic subgraphs to connect that makes the problem challenging.

Note that the query time reported, although impractical for real-time scenarios, refers to multi-exemplar exact answers. This points to the study of approximate schemes [6], [14] as an interesting direction for future studies.

Comparison to PANDA [14]: We compared the running time of PANDA and `mQ-Fast`$^+$ on *Cit-HepPh*. The

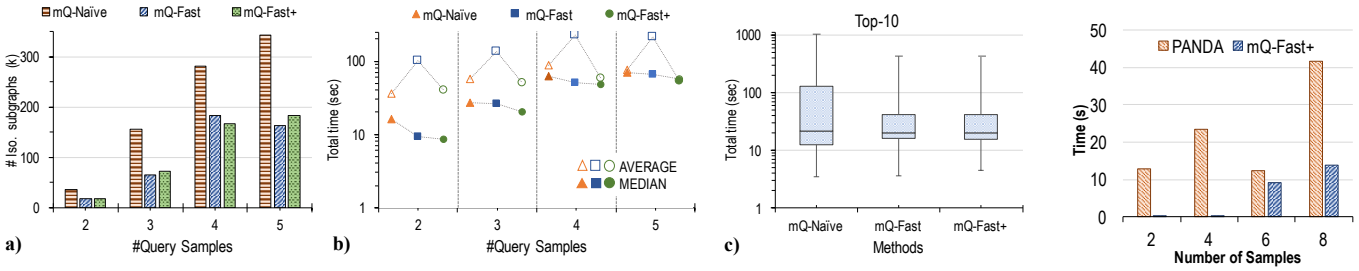

Fig. 4. (a) Median number of isomorphisms performed as a function of the number of query samples (top-3 answers, `Structural similarity`). (b) Average and median query time as a function of the number of samples in the query (top-3 answers, `Structural similarity`) (c) Comparison of Running Time on YAGO.

Fig. 5. Running Time Comparison of Fast+ vs. PANDA on *Cit-HepPh*.

experiment with queries containing between 2 to 8 samples (Figure 5) showed that `mQ-Fast`$^+$ is much faster for this task. We also tried to run queries on `YG` and `FB`, but since PANDA has to first build all isomorphic answers (similar to `mQ-Naive`) without employing any optimization for this task, the PANDA approach did not terminate within 1 hour, and we were not able to compare on larger datasets. Moreover, their approach does not enumerate all the answers but instead stops when finding portions of the graphs that contain them. Therefore, when considering their running time, the time needed for this extra step should be taken into consideration.

Expressive Power: Although an analysis of the quality of the algorithms largely depends on the choice of the weight function, we show some non-trivial result found with multi-exemplar queries. In this case, multi-exemplar queries can be employed for cinema journalism to quickly retrieve facts on actors and movies and their biographical information. For instance, in Figure 6 the samples describe notable actors, movies, and facts like prize won, spouse, or father-child relationships. We run this query on `YG`, which is not complete, so the samples are not part of the results since none of the examples have all the relationships required. Since none of the examples are part of an answer, with only this information as input any other query paradigms will fail. First we note that in `YG` the notion of *child* and *successor* are somehow collapsed, so that *George H.W. Bush* is listed as *child* of Ronald Reagan. Nonetheless, we find the 40th president of the U.S. has been an actor, and his wife as well acted in the same movie. We retrieve a similar case for *A. Schwarzenegger*, and one more for *Ronald Reagan*, now with the actual son *Ron*. Note how the results are nonisomorphic one another.

V. Related Work

By-example methods. Query-by-Example (QBE) [38] describes a query interface for relational databases by means of a template tuple, where the attribute values are partially specified by the user. Other approaches [13] accept tuples that should be included in the final desired result-set and the system infers the select-project-join queries that result in such tuples. Variations of this idea include the ability of the user to provide examples that are marked as relevant or irrelevant [7], or tuples alongside explanations [8]. Previous work on relational and textual data has no straightforward adaptation to more complex structures such as graphs, and assumes that the provided examples must appear entirely in the answers.

By-example works in graphs are divided into entity-based and structure-based. Entity-based approaches, like QBEES [9], take entities (nodes) as examples and return other similar entities. Structured-based approaches take as input more complex examples [6], [10]. Exemplar Queries [6] define a general paradigm for searching by-example and is applied on knowledge graphs. GQBE [10] instead considers tuples of entity mentions (such as ⟨Barack Obama, USA⟩) as input and finds other similar tuples. While in exemplar queries the user is able to provide complex structures, in GQBE only a list of entities forming a path is allowed as query. In this sense GQBE is a sub-problem of exemplar queries with a non-generic ranking function, and no solution for the case where the multiple examples in the input are only partial specifications. In contrast to the by-example works, in our work, the structure of the returned results does not need to be known in advance, and the input examples might not be part of the query answer.

Multiple queries on graphs. There are numerous methods [39] dealing with the optimization of single queries, but not for multiple small queries. A graph query can be seen as a multi-join query on the single edges. This has been considered especially in RDF databases [39]. The main limitation is that they require a fully specified query as input, which is not our case.

Other works consider the case of multiple query optimization in SPARQL queries [40]. SPARQL queries allow an optional part to be specified to generate queries with different structure. However, while in their case the number of options is limited, here we consider any possible structure combination in the results. The only solution would be to generate beforehand all the combinations as optional SPARQL parts, which is clearly impractical.

Finally, a recent work (PANDA) studies partial topology-based network search [14]. That is, to find the connections (paths) between structures node-label isomorphic to different user inputs. PANDA first materialize all isomorphic graphs, then groups them into connected com-

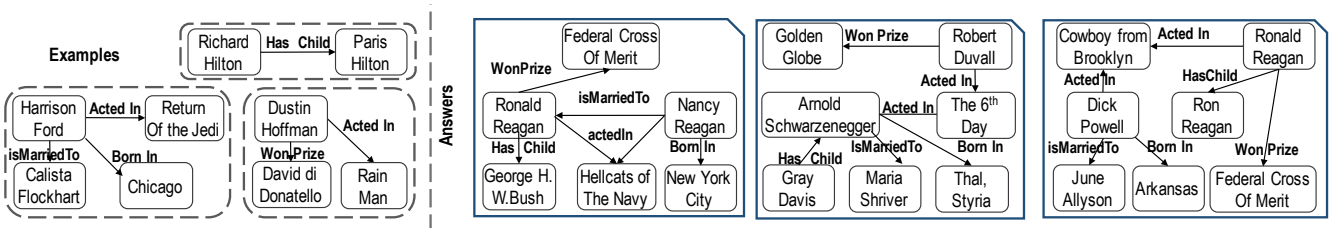

Fig. 6. Left: examples of actors and interesting biographical informations. Right: Answers using multi-exemplar paradigm.

ponents, and finally finds undirected shortest paths among them. Such semantics is mostly related to other solutions trying to find connections among disconnected nodes [22]. Moreover, as shown in the experiments (Section IV), the proposed exact solution does not scale to large graphs.

VI. Conclusions

By-example methods have been proven useful, but are limited to a single structure. In this work, we propose multi-exemplar queries, a novel query paradigm that identifies elements that are similar to a *set* of examples provided by the user, without enforcing the complete structure of the answer in advance. We describe efficient, exact solutions, and introduce a generic formulation to efficiently return top-k answers. The experiments show the efficiency and effectiveness of our approach. As future work, we plan to explore approximate solutions that can cater to online applications.

References

[1] H. V. Jagadish, A. Chapman, A. Elkiss, M. Jayapandian, Y. Li, A. Nandi, and C. Yu, "Making database systems usable," in *SIGMOD*, 2007.

[2] D. Mottin, M. Lissandrini, Y. Velegrakis, and T. Palpanas, "New trends on exploratory methods for data analytics," *PVLDB*, vol. 10, no. 12, pp. 1977–1980, 2017.

[3] G. Diaz, M. Arenas, and M. Benedikt, "SPARQLByE: Querying RDF data by example," *PVDLB*, vol. 9, no. 13, Sep. 2016.

[4] M. Zhu and Y.-F. B. Wu, "Search by multiple examples," in *WSDM*, 2014.

[5] D. Mottin, M. Lissandrini, Y. Velegrakis, and T. Palpanas, "Exemplar queries: Give me an example of what you need," *PVLDB*, vol. 7, no. 5, 2014.

[6] ——, "Exemplar queries: A new way of searching," *The VLDB Journal*, vol. 25, no. 6.

[7] K. Dimitriadou, O. Papaemmanouil, and Y. Diao, "Explore-by-example: An automatic query steering framework for interactive data exploration," in *SIGMOD*, 2014.

[8] D. Deutch and A. Gilad, "QPlain: Query by explanation," in *ICDE*, 2016.

[9] S. Metzger, R. Schenkel, and M. Sydow, "QBEES: query by entity examples," in *CIKM*, 2013.

[10] N. Jayaram, A. Khan, C. Li, X. Yan, and R. Elmasri, "Querying knowledge graphs by example entity tuples," *TKDE*, vol. 27, no. 10, 2015.

[11] M. Lissandrini, D. Mottin, D. Papadimitriou, T. Palpanas, and Y. Velegrakis, "Unleashing the power of information graphs," *SIGMOD Rec.*, 2014.

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

[13] F. Psallidas, B. Ding, K. Chakrabarti, and S. Chaudhuri, "S4: Top-k spreadsheet-style search for query discovery," in *SIGMOD*, 2015.

[14] M. Xie, S. S. Bhowmick, G. Cong, and Q. Wang, "PANDA: Toward partial topology-based search on large networks in a single machine," *VLDBJ*, vol. 26, no. 2, 2017.

[15] J. Cheng, X. Zeng, and J. X. Yu, "Top-k graph pattern matching over large graphs," in *ICDE*, 2013.

[16] R. De Virgilio, A. Maccioni, and R. Torlone, "R2G: a tool for migrating relations to graphs." in *EDBT*, 2014.

[17] M. Hacherouf, S. N. Bahloul, and C. Cruz, "Transforming XML documents to OWL ontologies: A survey," *JIS*, vol. 41, no. 2, 2015.

[18] A. Bordes and E. Gabrilovich, "Constructing and mining web-scale knowledge graphs: KDD 2014 tutorial," in *KDD*, 2014.

[19] S. R. Proulx, D. E. Promislow, and P. C. Phillips, "Network thinking in ecology and evolution," *TREE*, vol. 20, no. 6, 2005.

[20] H.-P. Kriegel, M. Renz, and M. Schubert, "Route skyline queries: A multi-preference path planning approach," in *ICDE*, 2010.

[21] M. Gupta, J. Gao, X. Yan, H. Cam, and J. Han, "Top-k interesting subgraph discovery in information networks," in *ICDE*, 2014.

[22] G. Kasneci, M. Ramanath, M. Sozio, F. M. Suchanek, and G. Weikum, "Star: Steiner-tree approximation in relationship graphs." in *ICDE*, 2009.

[23] M. Kargar and A. An, "Keyword search in graphs: Finding r-cliques," *PVLDB*, vol. 4, no. 10, 2011.

[24] J. R. Ullmann, "An algorithm for subgraph isomorphism," *J. ACM*, vol. 23, no. 1, 1976.

[25] W.-S. Han, J. Lee, and J.-H. Lee, "Turboiso: Towards ultra-fast and robust subgraph isomorphism search in large graph databases," in *SIGMOD*, 2013.

[26] S. A. Cook, "The complexity of theorem-proving procedures," in *STOC*, 1971, pp. 151–158.

[27] Google, "Freebase data dumps," https://developers.google.com/freebase/data, 2014.

[28] L. Getoor, B. Taskar, and D. Koller, "Selectivity estimation using probabilistic models," in *SIGMOD Record*, vol. 30, no. 2, 2001.

[29] A. Wagner, V. Bicer, T. Tran, and R. Studer, "Holistic and compact selectivity estimation for hybrid queries over RDF graphs," in *ISWC*, 2014.

[30] S. Yang, F. Han, Y. Wu, and X. Yan, "Fast top-k search in knowledge graphs," in *ICDE*, 2016.

[31] G. Ausiello, A. D'Atri, and M. Protasi, "Structure preserving reductions among convex optimization problems." *JCSS*, vol. 21, no. 1, 1980.

[32] A.-L. Barabási, "Scale-free networks: a decade and beyond," *science*, vol. 325, no. 5939, 2009.

[33] H. Bast, F. Baurle, B. Buchhold, and E. Haussmann, "Easy access to the freebase dataset," ser. WWW, 2014.

[34] T. Rebele, F. M. Suchanek, J. Hoffart, J. Biega, E. Kuzey, and G. Weikum, "YAGO: A multilingual knowledge base from wikipedia, wordnet, and geonames," in *ISWC*, 2016.

[35] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, 2014.

[36] J. Hauke and K. Tomasz, "Comparison of values of pearson's and spearman's correlation coefficients on the same sets of data," *Quaestiones Geographicae*, vol. 30, no. 2, 2011.

[37] S. Ma, Y. Cao, W. Fan, J. Huai, and T. Wo, "Strong simulation: Capturing topology in graph pattern matching," *TODS*, vol. 39, no. 1, 2014.

[38] M. M. Zloof, "Query by example," in *AFIPS NCC*, 1975, pp. 431–438.

[39] T. Neumann and G. Weikum, "Scalable join processing on very large RDF graphs," in *SIGMOD*, 2009.

[40] W. Le, A. Kementsietsidis, S. Duan, and F. Li, "Scalable multi-query optimization for SPARQL," in *ICDE*, 2012.