Recurring Retrieval Needs in Diverse and Dynamic Dataspaces: Issues and Reference Framework

Barbara Catania, Francesco De Fino, Giovanna Guerrini
University of Genoa, Italy
{firstname.lastname}@dibris.unige.it

ABSTRACT

Processing information requests over heterogeneous dataspaces is very challenging because aimed at guaranteeing user satisfaction with respect to conflicting requirements on result quality and response time. In [3], it has been argued that, in dynamic contexts preventing substantial user involvement in interpreting requests, information on similar requests recurring over time can be exploited during query processing.

In this paper, referring to a graph-based modeling of dataspaces and requests, we propose a preliminary approach in this direction centered on the enabling concept of *Profiled Graph Query Pattern* (PGQP) as an aggregation of information on past evaluations of a set of previously executed queries. The information maintained in PGQP is not query results, as in materialized queries, but can include different kinds of data and metadata.

1. INTRODUCTION

Motivations. The last years have been characterized by a tremendous growth of available information, coming from information sources that are highly heterogeneous in terms of structure, semantic richness, and quality. Sources can indeed contain unstructured data as well as data with well-defined structure, strongly correlated and semantically complex but relatively static data (e.g., Linked Open Data), highly dynamic user-generated data. This information is, however, a resource currently exploited much below its potential because of the difficulties for the users in accessing it. Users would like to find answers to complex requests expressing relationships among the entities of their interest, but they are able to specify requests only vaguely because they cannot reasonably know the format and structure of the data that encode the information relevant to them.

As an example, consider a smart city explorer, that is, a set of smart information services offered by a municipality to users that, e.g., retrieves information about attractions, points of interests, and shops. A user might, for instance, ask for the authors of the figurative artworks she is watching (i.e., they are located close to her position), together with information about other places where such authors are currently exposing. In this specific context, data may come from different datasets, are heterogeneous and very dynamic, as the user may quickly change her position or the environment itself (including data) might be different at different instants of time.

Processing complex requests on diverse and dynamic sources requires: (i) request interpretation; (ii) source selection; (iii) actual processing of the request on the relevant, and usually big in size, sources. The overall process is costly and, nevertheless, it may not guarantee user satisfaction on the obtained result since the request (i) could be incorrectly interpreted, (ii) could be processed on inaccurate, incomplete, unreliable data, or (iii) could require a processing time inadequate to its urgency.

To make processing time more adequate to the urgency of the request, approximate query processing approaches can be considered, by providing a fast answer at the cost of a lower accuracy [4]. Additionally, to improve the interpretation of the request, and related source selection, one possibility is to rely on user involvement. Novel paradigms for exploratory user-data interactions, that emphasize user context and interactivity with the goal of facilitating exploration, interpretation, retrieval, and assimilation of information, are being developed. Such solutions, however, do not seem adequate in dynamic contexts, characterized by urgent requests and by communication means hampering user interaction.

Other innovative approaches should therefore be devised, relying on different kinds of information for finding possibly approximate answers to complex information needs, even vaguely and imprecisely specified, operating on the full spectrum of relevant content in a dataspace of highly heterogeneous and poorly controlled sources. In [3], to overcome difficulties related to heterogeneity and dynamic nature, the exploitation of additional information, in terms of user profile and request context, data and processing quality, similar requests recurring over time, is suggested.

In this paper, we focus on *similar requests recurring*

over time for improving approximate processing of requests that we assume already interpreted and represented according to a given formalism. Recurring retrieval needs are very common in dynamic contexts, such as, for instance, during or after an exceptional event (environmental emergencies or flash mobbing initiatives), in the context of users belonging to the same community or that are in the same place, possibly at different times. The information needs are widespread among different users, because induced by the event, the interests of the community, and the place, respectively. We aim at taking advantage of the experience gained by prior processing in new searches for limiting interpretation errors and response time, thus reducing the possibility of producing unsatisfactory answers.

Recurring retrieval needs have been considered in query processing for a very long time in terms of *materialized views* (see, e.g., [7, 8]) The idea is to precompute the results of some recurring queries as materialized views, select some of such views (*view selection problem*) and re-use them (*view-based query processing*) for the execution of new requests. Unfortunately, the usage of materialized views in the contexts described above suffers of some problems: (i) views associate a result to a given query, however in the reference contexts we may be interested in associating other or additional information (e.g., the set of used data sources or, under pay-as-you-go integration approaches [6], query-to-data mappings); (ii) view updates are very frequent in dynamic environments, reducing the efficiency of the overall system; (iii) view-based query processing techniques usually rely on a precise semantics while heterogeneity tends to favour approximation-based approaches.

Alternative usages of the concept of recurring retrieval needs in query processing have been recently provided but, also in this case, precise query processing is taken into account. For example, in [13], common needs are used with the aim of expediting the processing of graph queries against a database of graphs, by reducing the number of subgraph isomorphism tests to be performed.

Contributions. This paper presents the preliminary results of an on-going work aiming at exploiting information about similar requests recurring over time in approximate query processing of requests executed over diverse and dynamic dataspaces. We refer to a graph-based representation of dataspaces, interpreted as a collection of (possibly) schemaless data sources, and on a graph-based language for modeling user retrieval needs, expressed as entity relationships queries over such dataspaces. We propose a framework for representing and managing (sets of) recurring user requests, and for exploiting them in approximate query processing. The proposed framework is based on the concept of *Profiled Graph Query Pattern* (PGQP). Like materialized views, each profiled graph query pattern corresponds to a kind of index value associated with information related to the past evaluation of the queries it represents; however, differently from materialized views, such information does not necessarily correspond to previously computed results, rather it ranges from query results to any other higher level information collected during query processing (e.g., query-to-data mappings). In specifying the framework, we identify the need for PGQP-aware approximate query processing techniques and PGQP management issues raised by the framework. Each instantiation of the framework will lead to the definition of a specific PGQP-aware query processing approach for a given set of information associated with PGQPs. Approximated results can be generated for two main reasons: due to the dynamic nature of the dataspace, information associated with PGQPs may change between two distinct executions of the same query; PGQPs are selected based on similarity criteria with respect to the query at hand.

The paper finally proposes a specific instantiation of the framework, in the context of which the various components and concepts are formalized, and algorithms for the most relevant tasks are specified. In this instantiation, we formally define PGQPs but we leave to future work the association of PGQPs with information collected in prior executions.

Paper organization. The proposed framework is presented in Section 2 and a specific instantiation of the framework is defined in Section 3. Section 4 concludes by discussing a number of open issues we are investigating in this work-in-progress.

2. PGQP-BASED FRAMEWORK

The framework we propose is based on a graph-based representation of dataspaces, interpreted as a collection of (possibly) schemaless data sources, and of user retrieval needs, expressed as entity relationships queries over such dataspaces [14]. Specifically, we assume to deal with data sources represented in terms of *partially specified graphs*, where some information, associated with nodes, may be unknown [2]. Similarly, each request is represented in terms of a graph, specifying entities and relationships of interest, which may contain node variables, in order to characterize information to be retrieved [2]. As an example, Figure 2 (a) presents a graph query Q corresponding to the information need "the authors of the figurative artworks the user is watching (i.e., they are located close to her position), together with such artworks, a biography of the authors, and information about places in Genoa where the authors are currently exposing their artworks". Notice that, since the user request is related to her current position, it needs to be interpreted and processed before such position changes.

For representing and managing (sets of) recurring user requests, the framework relies on the enabling concept of *Profiled Graph Query Pattern* (PGQP). Each PGQP corresponds to a graph, associated with data or metadata related to the past evaluation of the queries it represents. Such information does not necessarily correspond to previously computed results, thus, PGQPs do not necessarily correspond to materialized views. Rather, any kind of metadata collected during query processing, as the set of used data sources,

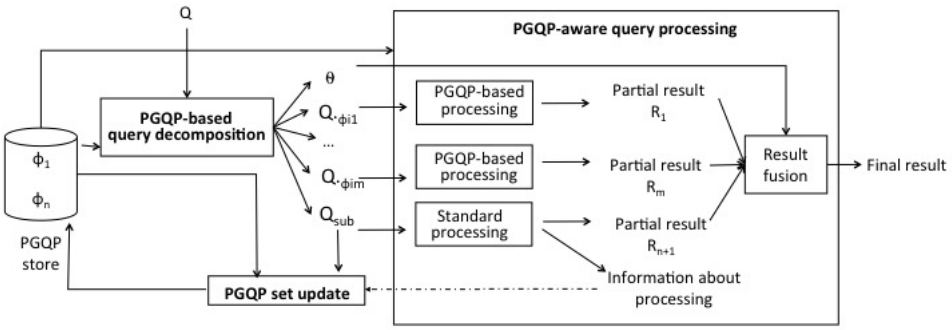

Figure 1: PGQP-based framework

query-to-data mappings under pay-as-you-go integration approaches [6], or result quality information, can be considered. As an example, Figure 2 presents a graph query Q and two PGQPs ϕ_1 and ϕ_2.

Independently from the information associated with PGQPs, the proposed framework aims at addressing three main problems which can be stated as follows (see Figure 1 for an overall picture).

PGQP-based query decomposition. Given a query Q and a set $\mathcal{S}$ of PGQPs, defined in terms of graph queries, determine whether it is possible to rely on $\mathcal{S}$ for executing Q. This is possible if Q can be approximated by a new query, obtained by composing a set of Q subgraphs, which best match PGQPs, with the portion of Q which cannot be represented in terms of $\mathcal{S}$. More formally, we look for a subgraph Q_{sub} of Q, a PGQP subset $\mathcal{S}_{\subseteq} \subseteq \mathcal{S}$, and a fusion operator θ such that Q can be rewritten in a new approximate query $Q_a = \theta(\{Q_{\phi_i} | \phi_i \in \mathcal{S}_{\subseteq}\}, Q_{sub})$, where Q_{ϕ_i} denotes the (sub)graph of Q for which an approximate match with (a subgraph of) ϕ_i exists, and Q_a satisfies some optimality criteria. That is, PGQPs in $\mathcal{S}_{\subseteq}$ are selected according to a similarity function δ which quantifies how close the query graph and each PGQP are.

With reference to Figure 2, various decompositions of Q are possible based on ϕ_1 and ϕ_2. Assuming ϕ_1 is more similar to Q than ϕ_2, the decomposition of Q can return: (i) $\mathcal{S}_{\subseteq} = \{\phi_1\}$; (ii) Q_{sub} defined as shown in Figure 2(a); (iii) θ defined as the join between the partial results of Q_{ϕ_1} (tuples for variables $?a$, $?m$, $?b$) and Q_{sub} (tuples for variables $?a$,$?b$). Notice that, by interpreting PGQPs as views, traditional view selection algorithms would not have selected any PGQP, due to the approximate matches between "Genoa" and "GE" and "Figurative" and "Figurative Art". Note that, even if in this example Q_{ϕ_1} totally matches ϕ_1, partial matches are also possible.

PGQP-aware query processing. In order to optimize Q_a evaluation, we can rely on information associated with PGQPs in $\mathcal{S}_{\subseteq}$. More precisely, each graph query Q_{ϕ_i} can be executed taking into account information related to its prior execution, that is, information associated with ϕ_i in $\mathcal{S}$. To this aim, specific PGQP-based processing algorithms should be designed, depending on the information associated with PGQPs. On the other hand, query Q_{sub}, which represents the residual part of Q, has to be executed based on a traditional graph query processing algorithm (see, e.g. [17]). At the end of the processing, all partial results are merged through the fusion operator θ and the result is returned to the user.

Referring to Figure 2(a), Q_{ϕ_1} is processed using information associated with ϕ_1 while Q_{sub} is processed by a traditional query processing algorithm.

PGQP set update. For each graph query Q, executed by the system without taking into account PGQPs, information related to its processing is collected (depending on the aim of the approach) and used, together with Q, for updating the set of available PGQPs at the end of the processing.

As an example, Figure 2(d) shows PGQP ϕ_1 extended with the part of the query executed by a traditional query processing algorithm.

The framework described above can be instantiated in several ways. As an example, consider an instantiation targeted to source selection. In this case, the information associated with PGQP ϕ_1 could be the sources exploited during the past processing of the queries it represents to retrieve the relevant information. Thus, for instance, information relevant for ϕ_1 may come from a dynamic data source s_1 containing information about art exhibitions of various artists (left part of ϕ_1) and a static data source s_2 (such as dbpedia) containing biographies of artists (right part). An important difference with respect to the materialized view approach is thus that the selected PGQP is not associated with query results (that would have required a frequent recomputation, given the dynamicity of data sources, e.g., in art exhibits) but the sources contributing data to the result. PGQP-based processing would rely on this information for targeting to these sources the execution of Q_{ϕ_1}. As an effect of the execution of Q, a third data source s_3 containing museum catalogues with artwork positions is identified, and associated with the updated PGQP (shown in Figure 2(d)).

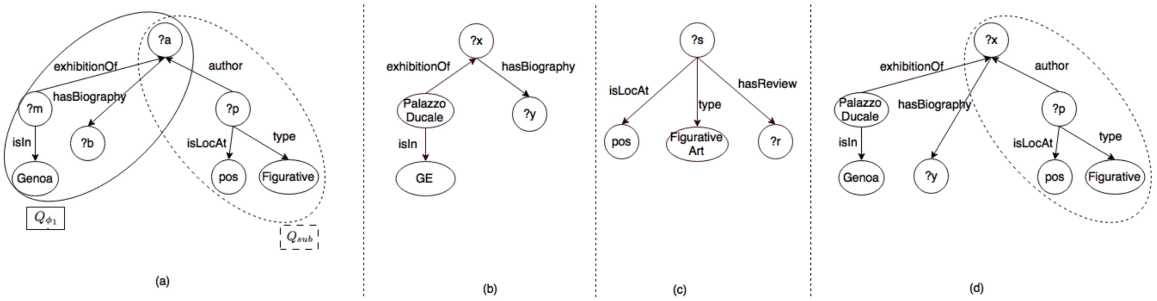

Figure 2: Example PGQP: (a) graph query Q; (b) PGQP ϕ_1; (c) PGQP ϕ_2; (d) PGQP ϕ_1 after the update phase

3. A PRELIMINARY PGQP-AWARE PROCESSING ALGORITHM

In this section, we select suitable models for dataspaces and user requests, we formalize the concept of PGQP and we discuss their use and management in a preliminar instantiation of the framework.

3.1 Dataspace and User Requests

Data Space Representation. Under the reference context, data spaces may contain redundant or even missing information. Furthermore, the heterogeneous nature of data leads to schemaless representations. To take care of these features, we assume the data space is represented in terms of a collection of *graph pattern databases*, i.e., graphs where nodes can also be labeled with variables, for representing situations in which the node identity is not known [1]).

DEFINITION 1. *Let $\mathcal{N}$ be a set of node labels. Let $\mathcal{V}_{node}$ be a set of node variables. Let Σ be an arbitrary (finite or infinite) set of symbols. A* Graph Pattern Database *over Σ is a pair $\pi = (N, E)$ where: (i) $N \subseteq \mathcal{N} \cup \mathcal{V}_{node}$ is the finite set of nodes; (ii) $E \subseteq N \times \Sigma \times N$ is the set of edges.* □

The semantics of a graph pattern database corresponds to all graphs which are instances of the pattern and it is defined via homomorphisms. Given a graph $G = (N', E')$ and a graph pattern database $\pi = (N, E)$, a homomorphism $h : \pi \to G$ is a mapping $h : N \to N'$ that maps labels and variables used in π to labels used in G such that:

1. $h_1(n) = n$, for every node label $n \in N$; and
2. for every edge $(n_1, e, n_2) \in E$, there is a path $(h(n_1), e, h(n_2))$ in G.

The semantics of a graph pattern w.r.t. the labeling alphabet Σ is $[[\pi]]_\Sigma = \{G \text{ over } \Sigma | G \models \pi\}$, where $G \models \pi$ holds if a homomorphism $h : \pi \to G$ exists.

Graph Query Language. In this work, for the sake of simplicity and for guaranteeing a tractable combined and data complexity (fundamental requirements for dealing with massive in size graph databases), we consider graph queries specified as *unions of acyclic conjunctive queries* and we denote with UACQ the resulting language [1, 2]. Formally, an UACQ query Q is the union of expressions $Q_1, ...Q_k$ where each Q_h, $1 \leq h \leq k$, is an expression of the form

$$ans(z_1, ..., z_n) \leftarrow \bigwedge_{1 \leq i \leq m} (x_i, a_i, y_i)$$

such that the graph underlying Q_h is acyclic, $m > 0$, each x_i and y_i is a node variable or a constant ($1 \leq i \leq m$), each $a_i \in \Sigma$ ($1 \leq i \leq m$), and each z_i is some variable x_j or y_j ($1 \leq i \leq n$, $1 \leq j \leq m$). The query Q is Boolean if $ans()$, i.e. $n = 0$.

UACQ queries can always be interpreted as graph (pattern) queries $Q = (\epsilon, \overline{z})$ where ϵ is the graph pattern underlying Q and $\overline{z} = (z_1, ..., z_n)$.

The semantics of a UACQ query $Q = (\epsilon, \overline{z})$ when executed over a graph pattern database $\pi = (N, E)$ can be defined as the set of answers which are returned for each instance of the graph pattern database. According to [2], we call them *certain answers* and we compute them as:

$$certain_\Sigma(Q, \pi) = \{\overline{v} \in N^n | \pi \models \epsilon[\overline{v}/\overline{z}]\}$$

where, given two graph patterns π_1 and π_2, $\pi_1 \models \pi_2$ (π_1 implies π_2) if $[[\pi_1]]_\Sigma \subseteq [[\pi_2]]_\Sigma$.

3.2 Profiled Graph Query Patterns

A PGQP represents in a compact way a set of graph queries executed in the past. In order to provide a formal definition of PGQPs, we propose to consider union as aggregate operator of the component queries and to represent a PGQP as a UACQ query. Since PGQPs are used for the approximate evaluation of queries, the issue also arises of determining whether each PGQP represents, in a reasonably good way, (a portion of) a graph query Q. If this is the case, ϕ becomes a candidate for the PGQP-aware processing of Q and we say that Q *matches* the PGQP. PGQP candidates can be selected through *approximate subgraph matching* [12] and the usage of a *graph similarity function.*

DEFINITION 2. *Let $Q_1 = (N_1, E_1), Q_2 = (N_2, E_2) \in$ UACQ. An Approximate Subgraph Match between Q_1 and Q_2 is a bijection mapping $\lambda : N_1' \leftrightarrow N_2'$, where $N_i' \subseteq N_i$, $i = 1, 2$. An Approximate Subgraph Matching Function is a function m that, for each pair of*

Algorithm 1 PGQP-based query decomposition

INPUT
Q: graph query
$\mathcal{S} = \{\phi_1, \phi_2, ..., \phi_n\}$: set of PGQPs
δ_m: a graph similarity function induced by an approximate subgraph matching function m
v_s: minimum allowed graph similarity
OUTPUT
$(\mathcal{S}_\subseteq, Q_{sub})$, such that $\mathcal{S}_\subseteq \subseteq \mathcal{S}$ and Q_{sub} is a subgraph of Q
APPROACH
1: Let $\phi_k \in \mathcal{S}$ such that $\delta_m(Q, \phi_k) = max_{1 \leq i \leq n} \delta_m(Q, \phi_i)$
2: **if** $\delta_m(Q, \phi_k) \geqslant v_s$ **then**
3: **return** $(\{Q_{\phi_k}\}, Q - Q_{\phi_k})$
4: **else**
5: **return** $(\{\}, Q)$

Algorithm 2 PGQP set update

INPUT
Q: graph query
$\mathcal{S} = \{\phi_1, \phi_2, ..., \phi_n\}$: set of PGQPs
v_c: maximum allowed transformation cost
OUTPUT
$\overline{\mathcal{S}}$: an updated set of PGQPs
APPROACH
1: Let $\phi_h \in S$ such that $\mathcal{C}_\mathcal{T}(Q, \phi_h) = min_{1 \leq i \leq n} \mathcal{C}_\mathcal{T}(Q, \phi_i)$
2: **if** $(\mathcal{C}_\mathcal{T}(Q, \phi_h) \leq v_c)$ **then**
3: $\overline{\phi_h} = \mathcal{T}(Q, \phi_h)$
4: $\overline{\mathcal{S}} = \mathcal{S} \setminus \{\phi_h\} \cup \{\overline{\phi_h}\}$
5: **else**
6: $\overline{\mathcal{S}} = \mathcal{S} \cup \{Q\}$
7: **return** $\overline{\mathcal{S}}$

$Q_1, Q_2 \in UACQ$, returns an approximate subgraph match from Q_1 to Q_2. A Graph Similarity Function *is a metric function $\delta : UACQ \times UACQ \rightarrow [0, 1]$ such that, for each $Q_1, Q_2 \in UACQ$, $\delta(Q_1, Q_2)$ quantifies the similarity between Q_1 and Q_2. δ is induced by an Approximate Subgraph Matching Function m (denoted by δ_m) if the value assigned to (Q_1, Q_2) quantifies the similarity between subgraphs Q_1' and Q_2' involved in the match $m(Q_1, Q_2)$.* □

PGQPs and PGQP matches can now be defined.

Definition 3. *A* Profiled Graph Query Pattern *ϕ is a graph query in UACQ. Let δ_m be a graph similarity function induced by an approximate subgraph matching function m. A graph query Q matches a PGQP ϕ, with similarity $v \in (0, 1]$, if $\delta_m(Q, \phi) = v$. Q_ϕ denotes the (sub)graph of Q involved in the match. Q is a* total match *of ϕ if $Q_\phi = Q$, it is a* partial match *otherwise.* □

In the following, we assume that function δ_m is provided by the system and used during PGQP-based query decomposition and PGQP set update.

3.3 PGQP-based Query Decomposition

In the following, we present a preliminary PGQP-based query decomposition approach, defined assuming to: (i) select only one PGQP (thus, $|\mathcal{S}_\subseteq| = 1$); (ii) merge partial results thought relational operators.

Several approximate graph matching functions have been proposed so far which can be used as basis for defining similarity functions (see, e.g., [9, 10] for some recent papers and [15] for a survey). A relevant example, due to its flexibility, is represented by the Similarity Flooding function [11]. Based on the intuition that similar nodes have similar neighbors, node similarity values are computed relying on information about node neighbors through an iterative fixpoint computation. The selection metric, used in Selection Flooding to identify the best mapping, can be used (after normalization of values between 0 and 1) as a graph similarity function. Details about PGQP definition in terms of the Similarity Flooding function can be found in [5].

According to Definition 3, each PGQP ϕ divides a graph query Q into two subqueries, namely Q_ϕ and $Q - Q_\phi$ (computed by removing from Q first edges in Q_ϕ and then all remaining non connected nodes belonging to Q_ϕ). This consideration guides us in the definition of a preliminary decomposition algorithm (see Algorithm 1). Given a query Q and a set of PGQPs $\mathcal{S}$, the PGQP ϕ_k providing the highest similarity value with respect to Q is selected and, if such value is greater than a given threshold (thus, Q and ϕ_k are close enough), the binary decomposition described above is returned. Otherwise, no decomposition is applied.

3.4 PGQP Set Update

In order to update the PGQP store, we rely on a *graph transformation function.*

Definition 4. *A* Graph Transformation Function *is a function $\mathcal{T} : UACQ \times UACQ \rightarrow UACQ$ such that, for each query Q and PGQP ϕ, Q is a match of $\mathcal{T}(Q, \phi)$, based on δ_m.* □

Transformation functions can be defined in terms of a small set of operations, previously defined in the context of the so called *tree editing problem* [16], which at least contains node relabelling, deletion and insertion. Each transformation function can be defined by specifying which operations should be applied to a given PGQP for each node and edge of the query Q at hand, in such a way that Q, after the transformation, matches the resulting PGQP. An example of graph transformation is the function that, given a PGQP ϕ and a query Q, inserts into ϕ all nodes and edges in $Q - Q_\phi$ that have not been mapped into ϕ nodes by δ_m.

In order to identify the best transformation, a *graph cost function* can be defined as follows.

Definition 5. *A* Graph Cost Function *for a transformation function $\mathcal{T}$ is a function $\mathcal{C}_\mathcal{T} : UACQ \times UACQ \rightarrow \mathbb{R}$, which, for each graph query Q and PGQP ϕ, assigns a cost value to $\mathcal{T}(Q, \phi)$.* □

Similarly to the tree editing problem, a graph cost function quantifies the transformation effort by assigning a cost to each applied editing operation. As an example, given a graph query Q and a PGQP ϕ, cost 1 can be assigned to the insertion of each new node and new edge in ϕ; a cost proportional to their distance can be assigned to each node n in Q which is mapped into a node v' in ϕ by δ_m.

Algorithm 2 presents a simple approach for PGQP set update based on the notions introduced above. In particular, it updates the input set of PGQPs by first selecting the PGQP leading to the lowest transformation cost for the input graph query Q. If such cost is lower than a given threshold, the PGQP set is updated by replacing the selected PGQP with the PGQP obtained by the transformation. Otherwise, the PGQP set is updated by inserting Q as a new PGQP.

EXAMPLE 1. *Consider the graph query Q and the PGQP set $\mathcal{S} = \{\phi_1, \phi_2\}$ presented in Figure 2. Consider the graph similarity function δ_m induced by the Similarity Flooding algorithm [11] (see [5] for details), a minimum similarity threshold $v_s = 0.7$, and a maximum cost threshold $v_c = 3$.*

Q is processed as follows. Algorithm 1 first computes the mapping and the similarity between Q and each ϕ_i, relying on δ_m. Partial approximate subgraph matches can be generated from Q to both ϕ_1 and ϕ_2. Assume the following graph similarities are computed: $\delta_m(Q, \phi_1) = 0.83$ and $\delta_m(Q, \phi_2) = 0.40$. PGQP ϕ_1 is selected because it is the most similar to Q. Since similarity is greater than v_s, decomposition is applied and the pair $(\{Q_{\phi_1}\}, Q - Q_{\phi_1})$ is returned for processing.

At the end of the processing, Algorithm 2 is used for updating the PGQP store. Suppose that $\mathcal{C}_\mathcal{T}(Q - Q_{\phi_1}, \phi_1) = 2.56$ and $\mathcal{C}_\mathcal{T}(Q - Q_{\phi_2}, \phi_2) = 3$. PGQP ϕ_1 is selected because of the lowest transformation cost. Since the transformation cost is lower than v_c, PGQP ϕ_1 is updated by inserting all nodes and edges in $Q - Q_{\phi_1}$ not mapped into ϕ_1 nodes by δ_m. ◇

4. OPEN ISSUES

The work reported in this paper represents just the first step towards the design of approximate query processing approaches for recurring retrieval needs. Several issues still need to be investigated. Some of them are discussed in what follows.

Framework instantiation. In this paper, we considered a simple notion of approximate subgraph matching, defined by relaxing the notion of subgraph isomorphism. Other approaches can however be considered, which allow more flexible types of approximation (see, e.g., [15] for a survey). More effective query decomposition approaches should also be designed which increase the number of generated subqueries, together with specific optimality criterias; optimization approaches should be developed which take into account the number and the shape of the identified subqueries for selecting the most efficient PGQP-based execution plan.

Information associated with PGQPs. As already discussed, several types of information can be associated with PGQPs. For each type of information, specific PGQP-based query processing algorithms should be devised.

PGQP management. In this paper, we assumed to deal with an initial set of PGQPs and we only preliminarily discuss how to modify it taking into account a new graph query execution. There are however several further issues to be taken into account for PGQP management, e.g., how to decide that a query is recurrent and should be indeed used for updating the PGQP set? What about the tradeoff between the size of the PGQP set, the query processing cost, and the effectiveness (in terms of result quality) of the overall query approach?

5. REFERENCES

[1] P. Barceló. Querying graph databases. In *Proc. of PODS*, pp. 175–188. ACM, 2013.

[2] P. Barceló, L. Libkin, and J. L. Reutter. Querying graph patterns. In *Proc. of PODS*, pp. 199–210. ACM, 2011.

[3] B. Catania et Al. Wearable queries: adapting common retrieval needs to data and users. In *Proc. DBRank Workshop*. ACM, 2013.

[4] B. Catania and G Guerrini. Approximate queries with adaptive processing. Advanced Query Processing, Intelligent Systems Reference Library (36), pp. 237-269. Springer, 2013.

[5] B. Catania, F. De Fino, and G. Guerrini. Recurring retrieval needs in diverse and dynamic dataspaces: issues and reference framework, extended version. Technical Report, University of Genoa, Italy, 2016.

[6] A. Das Sarma, X. Dong, and A. Halevy. Bootstrapping Pay-As-You-Go Data integration systems.In *Proc. of SIGMOD*. ACM, 2008.

[7] W. Fan, X. Wang, and Y. Wu. Answering pattern queries using views. *IEEE Trans. Knowl. Data Eng.*, 28(2):326–341, 2016.

[8] F. Goasdoué et Al. View selection in semantic web databases. PVLDB, 5(2): 97–108, 2011.

[9] J. He et Al. Assessing single-pair similarity over graphs by aggregating first-meeting probabilities. Inf. Syst. 42: 107-122, 2014.

[10] D. Koutra et Al. DeltaCon: Principled Massive-Graph Similarity Function with Attribution. TKDD 10(3): 28, 2016

[11] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proc. of ICDE*, pp. 117–128. IEEE, 2002.

[12] Y. Tian and J.M. Patel. TALE: A Tool for Approximate Large Graph Matching. In *Proc. of ICDE*, pp. 963-972, 2008

[13] J. Wang, N. Ntarmos, and P. Triantafillou. Indexing query graphs to speedup graph query processing. In *Proc. of EDBT*, pages 41–52. 2016.

[14] G. Weikum. Data and knowledge discovery. GRDI 2020, 2011.

[15] Y. Wu. Extending graph homomorphism and simulation for real life graph matching. PhD Thesis, 2010.

[16] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262, 1989.

[17] P. Zhao, J. Han. On graph query optimization in large networks. *PVLDB*, 3(1): 340-351, 2010.