Exploiting Recurrent Retrieval Needs in Querying Heterogeneous and Dynamic Graph Dataspaces

Discussion paper

Barbara Catania, Francesco De Fino, Giovanna Guerrini

University of Genoa, Italy
{firstname.lastname}@dibris.unige.it

Abstract. Querying data sources containing heterogeneous and dynamic information is a complex task: high quality answers need to be produced in a limited response time. Dynamic contexts preclude user involvement in interpreting the request and thus solutions should be devised, relying either on additional knowledge about the current execution (e.g., query context or user profile) or previous executions of the same request [**?**]. The aim of this discussion paper is to exploit similar requests recurring over time for improving query processing, in terms of result quality and processing time, and to provide a general framework for representing and managing information collected in the executions of (sets of) recurring queries. Our approach relies on the enabling concept of *Profiled Graph Query Pattern* (PGQP), which represents a set of (previously executed) queries associated with information about their executions. Differently from apparently similar proposals (materialized views, smart caching), the proposed approach approximately matches queries with profiled patterns with the aim of retrieving various kinds of information, related to past executions of similar queries, and improving the processing of the query at hand.

1 Introduction

The last few years have been characterized by the growth of highly heterogeneous and dynamic data sources, shared across the network. These sources usually contain both structured and unstructured data, strongly correlated and often highly dynamic. Such data are often exploited much below their potential due to difficulties in accessing them. Indeed, users can specify their request only vaguely because the format and the structure of the data encoding the relevant information is unknown. As an example, consider a smart city explorer, that is, a set of information published by a municipality to users that may retrieve, e.g., information about attractions, points of interests, and shops. In this specific context, data may come from different, heterogeneous and very dynamic datasets. At different instants of time, also user position or the environment itself (including data) may be changed.

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. Besides being costly, the overall process may not guarantee the user is satisfied by the obtained result due to various problems: the request could be incorrectly interpreted or processed on inaccurate, incomplete, unreliable data; additionally, processing time could be inadequate to the query urgency. A common solution to reduce query processing time is to rely on approximate processing approaches, which provide a faster answer at the cost of a lower accuracy. On the other hand, user involvement in the interpretation of the request [**?**], is not adequate in dynamic contexts, where urgent requests hamper user interaction. In [**?**], we claimed that, to cope with the difficulties raised by the heterogeneity and dynamic nature of the considered environments, user profiles and request contexts, data and processing quality, and similar requests recurring over time could be exploited.

In this paper, we focus on *similar requests recurring over time* for improving approximate processing of requests, assumed to be already interpreted and represented according to a graph-based formalism [**?**]. Recurring retrieval needs have been considered in query processing for a very long time in terms of *materialized views* [**?**,**?**]. 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 from some problems: (i) views associate a result with a given query, however in the reference contexts other or additional information could be of interest (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. Recurring retrieval needs are also at the basis of *smart caching* approaches. Here the main issue concerns the definition of suitable cache replacement policies. Some of the proposed approaches rely on approximate query matching [**?**,**?**] but, similarly to materialized views, cached queries are usually associated with query results.

The framework we propose aims at representing and managing recurring queries in order to improve query processing, in terms of result quality and processing time, getting over the limitations of current materialized views and smart caching approaches by relying on the enabling concept of *Profiled Graph Query Pattern* (PGQP). The idea is to take advantage of prior processing in order to obtain *shortcuts* to different points in the query processing stack. The approach we provide generalizes smart caching methods allowing various kinds of information to be associated with cached queries in order to improve query processing even further.

The remainder of the paper is organized as follows. PGQPs are introduced in Section **??**. The main phases of the proposed framework are presented in Sections **??** and **??**. Finally, Section **??** concludes by outlining some future developments.

2 Profiled Graph Query Patterns

The framework we propose is based on a graph-based representation of data spaces and user retrieval needs [**?**]. In particular, 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 assume the data space is represented in terms of a set of *graph data sources*, i.e., graphs where nodes can also be labeled with variables, for representing situations in which the node identity is not known [**?**], due to the heterogeneous nature of data and the possible lack of information. Furthermore, we assume a graph query is specified as a directed, labeled graph with variables belonging to the set of *Unions of Acyclic Conjunctive Queries* (UACQ) [**?**].

An example of a UACQ query is shown in Figure **??**(a). The query represents the following request: find 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. Since the user request is related to her current position, it needs to be interpreted and processed before such position changes. Additionally, data sources, and, in particular, information about exhibitions are dynamic due to frequent exhibit schedule updates.

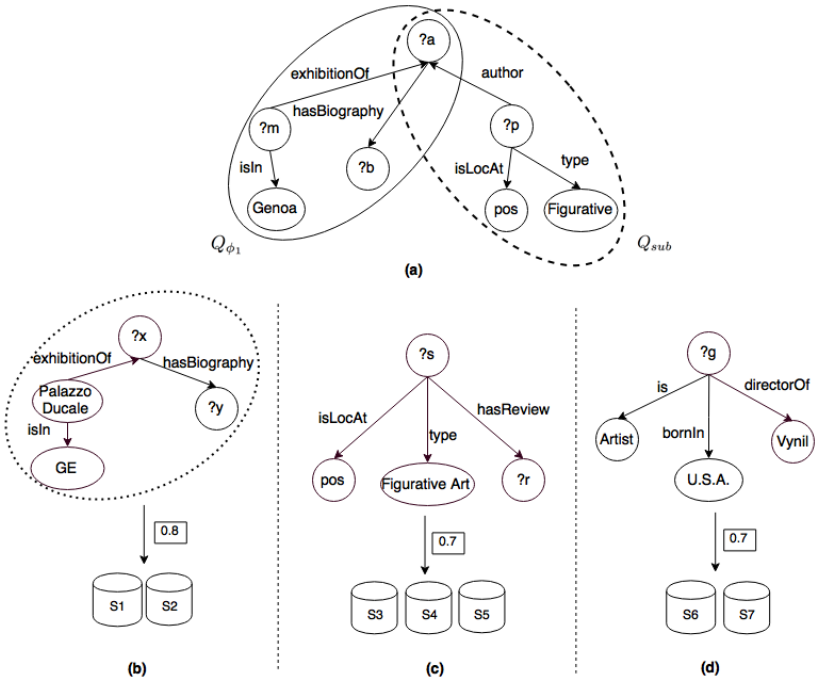

Fig. 1. (a) Graph query Q; (b) PGQP ϕ_1; (c) PGQP ϕ_2; (d) PGQP ϕ_3

For representing and managing (sets of) recurring user requests, the proposed framework relies on the enabling concept of *Profiled Graph Query Pattern* (PGQP), introduced in [**?**]. Each PGQP corresponds to a graph query, associated with data or metadata related to its past processing, together with some values quantifying the accuracy of such association. More formally, a PGQP ϕ can be defined as a triple $< Q, \mathcal{I}, v >$ where $Q \in UACQ$, $\mathcal{I}$ is a set of processing information, generated during the past processing of Q over the data space, and $v \in [0, 1]$ is a value which quantifies the accuracy of associating Q with I (we will detail this concept later).

Several kinds of information can be associated with PGQPs; when previously computed results are associated with a graph query, a PGQP becomes a slight extension of a materialized view. In general, however, any kind of metadata collected during query processing (e.g., the set of used data sources, query-to-data mappings under pay-as-you-go integration approaches [**?**]) can be considered. Any specific instantiation of the general definition makes PGQPs a shortcut to different points of the query processing stack.

Example 1. Consider an instantiation of the PGQP concept defined above targeted to source selection and assume that each PGQP graph query is associated with the specific data sources exploited during its past processing. Figure **??**(b),(c),(d) shows some examples of PGQPs. Specifically, PGQP ϕ_1 represents the following request, represented as a UACQ query: find the authors ($?x$) which are exposing at Palazzo Ducale in GE (which is an acronym for Genoa), together with a biography ($?x$) of such authors. Such query is associated with sources S_1 and S_2, meaning that past executions of the PGQP query relied on such sources for computing the query result. Data source S_1 may correspond to a dynamic data source containing information about art exhibitions of various artists (leftmost part of updated ϕ_1); on the other hand, S_2 may correspond to a static data source S_3 (such as dbpedia) containing biographies of artists. Value 0.8 is the accuracy measure associated with ϕ_1 which specifies that the results computed during past executions relying on data sources S_1 and S_2 could be considered 80% accurate. 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 with the sources contributing data to the result. Additionally, even by associating query results to PGQPs, graph ϕ_1 interpreted as a view would not have been selected since (referring for instance to the approach in [**?**]) the match between "Genoa" and "GE" is approximate and, thus, graph ϕ_1 cannot contribute to compute Q results. $\diamond$

Figure **??** summarizes the overall PGQP-based framework we envision, assuming a set of PGQPs is available. It relies on two main phases: (i) the aim of the first phase (PGQP-based Query Processing) is to decompose a given query into a set of sub-queries which could be executed by relying on information association with PGQPs, execute them, and compute the final results. For each sub-query, an accuracy value should also be computed that quantifies how good is any generated partial result with respect to the original request; (ii) the aim of

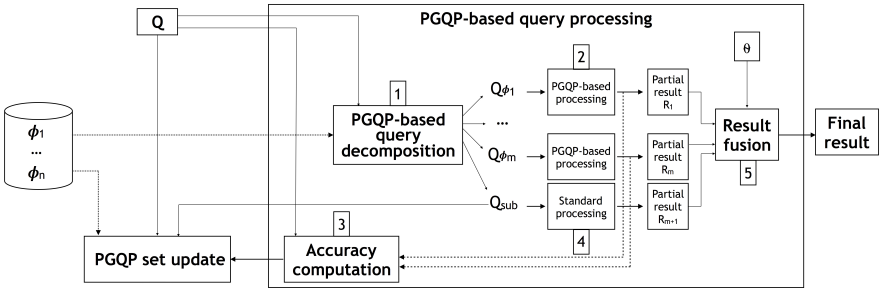

Fig. 2. PGQP-aware framework

the second phase (PGQP Set Management) is to refresh the PGQP set, possibly taking into account new information generated during query processing.

3 PGQP-Based Query Processing

Given a query Q, PGQP-based query processing relies on information associated with PGQPs to rewrite Q into an equivalent one and to process it improving evaluation performance and result quality. Six main steps can be identified.

1. PGQP-based query decomposition. Given a query Q and a set $\mathcal{S}$ of PGQPs, defined in terms of graph queries, we first 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, denoting with $\phi_i.1$ the query represented by the PGQP, 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) $\phi_i.1$ exists, and Q_a satisfies some optimality criteria. PGQPs in $\mathcal{S}_{\subseteq}$ are selected by combining approaches for *approximate supergraph matching* and graph view selection. PGQPs ϕ_i in $\mathcal{S}_{\subseteq}$ are selected according to a ranking value. Such value could be generated by combining the similarity between Q and $\phi_i.1$, computed through the similarity function δ underlying the matching process and quantifying how close Q and $\phi_i.1$ are, and the PGQP accuracy measure $\phi_i.3$, according to specific functions.

2. PGQP-based processing. Each graph query Q_{ϕ_i} can be executed taking into account information related to its prior executions, that is, information associated with the query represented by ϕ_i in $\mathcal{S}$. To this aim, specific PGQP-based processing algorithms should be designed, depending on the information associated with PGQPs.

3. Accuracy computation. The evaluation of each Q_{ϕ_i} subquery returns in general an approximate result, for which an accuracy value should be computed. This value is influenced by three main components: (i) the similarity between Q_{ϕ_i} and $\phi_i.1$; (ii) the usage of either a precise or an approximate PGQP-based query processing algorithm; (iii) result cardinality, in case we want to deal also with the empty or few answers problem [**?**], since it may impact user satisfaction. The accuracy values associated with the result obtained by the processing of each Q_{ϕ_i} are then used to update the accuracy values associated with $\phi_i.1$. See Section **??** for additional details.

4. Residual query execution. 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. [**?**]). Processing information is collected during the execution and also, in this case, an accuracy measure has to be associated with the result. In case a precise query processing algorithm is used, such measure will be equal to 1.

5. Partial result merging. At the end of the processing, all partial results derived from the execution of each Q_{ϕ_i} and from the evaluation of the residual query Q_{sub} are merged through a fusion operator θ and the final result is returned to the user.

Example 2. Consider again Figure **??** and assume that $\delta(Q, \phi_1.1) = \delta(Q, \phi_2.1) = 0.8 > \delta(Q, \phi_3.1) = 0.4$. Suppose we want to select just one PGQP and that the ranking value $r(\phi)$, associated with each PGQP ϕ_i, is computed by multiplying the similarity and the accuracy values. In this case, PGQP ϕ_1 is selected for the PGQP-based processing, since $r(\phi_1) = 0.64, r(\phi_2) = 0.56$, and $r(\phi_3) = 0.28$. Thus, the decomposition step 1 will return: (i) $\mathcal{S}_{\subseteq} = \{\phi_1\}$; (ii) Q_{ϕ_1} defined as shown in Figure **??**(a); (iii) Q_{sub} defined as shown in Figure **??**(a); (iv) 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 $?a$, $?p$).

Suppose that the evaluation of Q_{ϕ_1} in step 2 generates the result R_1: {$?a$ = "Andy Warhol", $?m$ = "Palazzo Ducale", $?b$ = "Bio"}. In step 3, an accuracy value is computed for the result R_1, say 0.9. The evaluation of Q_{sub} in step 4 returns another partial result R_2, for example R_2: {$?a$ = "Andy Warhol", $?p$ = "Shot Marilyn"}. Finally, in step 5, R_1 and R_2 are joined, generating the final result {$?a$ = "Andy Warhol", $?m$ = "Palazzo Ducale", $?b$ = "Bio", $?p$ = "Shot Marilyn"} . ◇

4 PGQP Set Update

The PGQP-based query processing of a query produces various information that need to be reflected into the PGQP set to keep it up-to-date. More precisely, we envision two main groups of updates, namely *online updates*, executed as side effects of query processing, and *offline updates*, which periodically delete or refresh PGQPs which are considered unreliable.

Online Updates. PGQP-based query processing may generate two different types of information: (i) in step 3, an accuracy value is computed for each subquery executed by relying on a PGQP ϕ_i; such value can be used to update the aggregate accuracy value associated with ϕ_i; suitable aggregation measures should be defined to this purpose and are current under investigation; (ii) in step 5, new processing information $\mathcal{I}$ are generated from the processing of the residual query Q_{sub}, together with an accuracy value v; such information can be interpreted as a new PGQP $\phi = (Q_{sub}, \mathcal{I}, v)$, which should be used for updating the PGQP set in case Q_{sub} is assumed to be recurrent. Various approaches can be exploited to detect recurrent queries. A simple and optimistic approach could be that of assuming any new query is recurrent, then monitoring the usage of the related PGQP to refine such decision. If Q_{sub} is recurrent, ϕ should be added to the set and made available for future query processing.

Offline Updates. Due to dynamicity of the environment, PGQPs, after their creation, may become unreliable for three main reasons: (i) the query they represent may be no more considered as recurrent (see above); (ii) the accuracy value becomes very low, meaning that the usage of the PGQP lead to the generation of inaccurate results; (iii) the associated information may become imprecise due to the variability of the data space. For this reason, periodically the PGQP set should be refreshed. Different policies could be provided. As an example, any PGQP with an accuracy value lower than a given threshold could be considered unreliable and could be refreshed by executing the PGQP query, using a standard graph query processing approach: similarly to the execution of a residual query, new information gathered during query processing, as well as the computed result accuracy value, could be used to update the PGQP considered unreliable. Additionally, all PGQPs could be periodically refreshed to take care of data source dynamicity.

Example 3. Consider again query Q in Figure **??**(a). Based on what we stated in Example **??**, we know that, after the PGQP-based processing of Q_{ϕ_1}, an accuracy value equal to 0.9 is computed. This value is used to perform an online update with the aim of updating the accuracy value associated with Q_{ϕ_1}, taking into account this further PGQP-based execution. For example, the two values could be multiplied (since they can be interpreted as the probability of two independent events). Additionally, after the execution of the residual query Q_{sub}, PGQP $< Q_{sub}, \{S_8, S_9\}, 0.75 >$ could be added to the PGQP set, assuming it is recurring. Now assume that the resulting accuracy value is lower than a given threshold: this means that the processing information represented inside the PGQP led to inaccurate query results. In this case, query $\phi_1.1$ could be re-executed and the information collected during query processing used to refresh ϕ_1. For example, a new data source S_3 could be identified as relevant, while a previously identified data source, e.g., S_2, may become useless. $\diamond$

5 Discussion

This paper presents a first step towards the design of query processing approaches for recurring retrieval needs. Several issues still need to be investigated, concerning specific framework instantiations, characterization of specific information to be associated with graph queries in PGQPs and related PGQP-based query processing approaches, and suitable approaches for detecting recurring queries and refreshing the PGQP set. We are currently working on an instantiation of the framework tailored to source selection for linked open data. The idea is to associate each PGQP with information about the data sources used in the past for query execution (as discussed in the presented examples), thus providing an alternative approach with respect to index-based source selection for linked open data [**?**] in all contexts in which queries are recurring.

References

1. Agrawal, S., Chaudhuri, S., Das, G. and Gionis, A. Automated ranking of database query results. In CIDR, 2003.
2. Barceló, P. Querying graph databases. In *Proc. of PODS*, pp. 175–188, 2013.
3. Catania, B., De Fino, F. and Guerrini, G. Recurring retrieval needs in diverse and dynamic dataspaces: issues and reference framework. In *GraphQ Workshop*, 2017.
4. Catania, B., Guerrini, G., Belussi, A., Mandreoli, F., Martoglia, R., and Penzo, W. Wearable queries: adapting common retrieval needs to data and users. In *DBRank Workshop*, 2013.
5. Das Sarma, A., Dong, X. and Halevy, A. Bootstrapping Pay-As-You-Go data integration systems. In *Proc. of SIGMOD*, 2008.
6. Fan, W., Wang, X. and Wu, Y. Answering pattern queries using views. *IEEE Trans. Knowl. Data Eng.*, 28(2):326–341, 2016.
7. Goasdoué, F., Karanasos, K., Leblay, J., and Manolescu, I. View selection in semantic web databases. *PVLDB*, 5(2): 97–108, 2011.
8. O.Hartig and M. T. Özsu. Linked Data query processing. In *Proc. of ICDE*, pp. 1286–1289, 2014.
9. Mass, Y., Ramanath, M., Sagiv, Y., and Weikum, G. IQ: the case for iterative querying for knowledge. In *Proc. of CIDR* , pp. 38-44, 2011.
10. Melnik, S., Garcia-Molina, H. and Rahm, E. Similarity flooding: a versatile graph matching algorithm and its application to schema matching. In *Proc. of ICDE*, pp. 117–128, 2002.
11. Ntarmos, N., Triantafillou, P., and Wang, J. GraphCache: a caching system for graph queries. In *Proc. of EDBT*, pp. 13-24, 2017
12. Papailiou, N., Tsoumakos, D., Karras, P., and Koziris, N. Graph-aware, workload-adaptive sparql query caching. In *Proc. of SIGMOD*, pp. 1777-1792, 2015.
13. Tian, Y. and Patel, J.M. TALE: a tool for approximate large graph matching. In *Proc. of ICDE*, pp. 963-972, 2008.
14. Zhang, K. and Shasha, D. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262, 1989.
15. Zhao, P. and Han, J. On graph query optimization in large networks. *PVLDB*, 3(1): 340-351, 2010.
16. Weikum, G. Data and knowledge discovery. GRDI 2020, 2011.