Static and provenance analysis for safe and efficient query evaluation on large data graphs.

PhD project proposal

Dario Colazzo
LAMSADE - Univertité Paris Dauphine
dario.colazzo@dauphine.fr

Khalid Belhajjame,
LAMSADE - Univertité Paris Dauphine
khalid.belhajjame@dauphine.fr

Context

In recent years the database research community is dedicating more and more attention to the problems related to the efficient management of graph data. The interest in this kind of data is motivated by the widespread diffusion of applications that manipulate and process data that are naturally modelled as graphs: the most notable example of these application is represented by social networks like Facebook or LinkedIn, that daily process graphs with billions of edges, but graph data are heavily used also in application contexts like traffic analysis, crime detection and prevention, and bioinformatics.

One of the models for graph data that is gaining more and more interests in both the industrial and the research context is that of data graphs. Data graphs are directed, labeled graphs having attributes associated to vertices and/or edges. This model is general enough to capture RDF property graphs and to be used for integrating graph data with existing relational databases: for example, in crime prevention applications it is common to integrate data collected by national security agencies and/or police forces, e.g., criminal records, with data extracted from social networks, mails, text messages, and phone calls, so to identify possible threats to the national security and to dismantle terrorist cells.

In many cases data graphs are directly stored as plain or compressed text files in the file system; only in a few cases, input graphs are stored as edge collections in NoSQL systems or in traditional relational systems. To be queried and/or analyzed, data graphs must be imported and loaded in a graph processing system [4] or in a graph database management system (GDBMS). In the first class of systems, modeled around Google's Pregel [7], graph vertices run a user provided code and exchange messages in a synchronous or asynchronous way; in the latter class, the system provides the user a query language usually built around a subset of regular path queries. In both these classes, the loading phase usually involves the batch scan of the whole dataset, so to build the internal representation of the graph and/or any support data structure.

While there are some relevant graph applications that actually need to be executed on the whole input dataset, in other cases (e.g., regular path queries) only a limited subgraph is involved in the computation. In these contexts, the need to read and import the whole dataset may have some unpleasant consequences. First of all, this import phase can be very expensive, in particular in the very common context of big data graphs, and may account for a very large fraction of the time required by the user application to complete. Furthermore, in most graph processing systems graphs are kept in main memory, hence the need to keep in memory unnecessary graph fragments can significantly limit the scalability of the system. Finally, current GDBMSs do not employ any static analysis in order to check that structural constraints expressed by the queries correctly correspond to structural properties featured by the graph, hence queries are always evaluated, even though their result can be statically deemed as empty.

Another limitation of current approaches for querying large data graphs is the total lack of techniques to track the provenance of query results. Provenance analysis has been largely studied in the context of relational and XML query languages [5]. In the context of querying large graph databases
provenance analysis may provide useful information to explain to the user how a piece of result has been computed and from which parts of the input database it depends on.

Project

In order to overcome the above depicted limitations of current frameworks for managing large collections of graph data, in this PhD project we aim at devising and implementing techniques for:

- Inferring a data summary of the graphs, which provides a succinct and precise description of structural- and value-based properties met by the graph, and which is particularly tailored for data projection purposes.
- Statically analysing queries expressed by means of advanced graph query languages in order to match them against the inferred graph summary and inferring a projector graph summary (pgs) describing the part of the graph really needed by the query.
- Efficiently projecting the input graph by means of the pgs in order to reduce the size of the graph to be imported by GDBMS. This will allow the GDBMS to import only data relevant to the queries to be run.

We will develop the above techniques first in a centralised fashion, assuming that the size of input graphs is not too big, and then switch to the big data context where centralised approaches are unfeasible. To ensure reasonable processing time for data summary inference and projection, in this context we aim at devising new algorithms expressed according to paradigms in the MapReduce ecosystem, and based on techniques developed in the centralised setting.

In a second part of this PhD project we aim at studying provenance techniques for graph queries in two particular contexts. That of path-based querying \[\text{[8, 10]},\] at the basis of current, widely used graph query languages like Gremlin \[\text{[2]},\] and Cypher \[\text{[3]},\] and that of RDF querying in the presence of entailment (Sparql 1.1) \[\text{[6]},\] We will focus on where and how provenance, with a particular focus on slicing and tracing analysis \[\text{[9]},\] in order to provide the user with useful information about: i) which parts of the database have contributed to a given part of query results and ii) how some specific parts of a query have contributed to a fragment of query result. The use of such information can be crucial in various important tasks in the context of large graph querying, like confidence computation, view maintenance and update, and query debugging.

References


