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

PhD project proposal

Dario Colazzo
LAMSADE - Université Paris Dauphine
dario.colazzo@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 applications 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 or in a graph database management system (GDBMS). In the first class of systems, modeled around Google’s Pregel, 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.

Another limitation of current approaches for querying large data graphs is that techniques to track and analyse the provenance of query results are scarcely investigated, while provenance analysis has been largely studied in the context of relational and XML query languages. The use of information derived by provenance analysis can be crucial in various important tasks in the context of large graph querying, such as query debugging and view maintenance.
**Project**

In order to overcome the above depicted limitations, in the first part of this PhD project we aim at devising and implementing: i) techniques for inferring structural- and content-based properties from big graphs, and ii) static analysis techniques over inferred properties enabling optimisation in terms of memory usage and query execution time for existing graph processing engines.

In a second part of this PhD project we aim at studying provenance techniques for graph queries, with a focus on approaches able to provide explanations about how given parts of a query result has been calculated.

In order to ensure reasonable efficiency, in the context of both the above parts we will devise and implement algorithms according to paradigms in the MapReduce ecosystem. Particular emphasis will be given on the experimental analysis of devised algorithms.

**References**


