Graph Models & Database Management Systems

Graphs are semantically rich data models able to inherently capture the structure of complex objects and their interconnecting relationships. Due to their high expressiveness, graphs are used in numerous domains, including Knowledge Representation and the Semantic Web, Linked Open Data, geolocation data, as well as in life science repositories. Graph models cover the spectrum from simple, edge-labeled to property enriched, on both nodes and edges, and lie at the foundation of modern graph database systems. These systems leverage native graph storage and index-free adjacency to replace the costly table joins from the relational setting with efficient graph traversals. We overview recent work on computing compact abstractions of graph datasets and on using these to approximate the result of intractable path query fragments ([1-4]). We also highlight the implementation and formal verification of a prototype graph database engine capable of evaluating regular queries. To this end, we rely on a fragment of the Datalog logic programming language as a unifying formalism for the multitude of graph query dialects that currently underpin commercial implementations and that are yet to be standardized ([5-6]).

Approximate Graph Querying (AQP)

Approximate Graph Querying (AQP) is an essential feature for modern graph database engines to deal with the computational complexity of graph queries. It involves approximating the result of a complex query to make it feasible to process. This can be achieved through various techniques such as parameterized queries, sketches, or sampling.

GRASP Summaries

GRASP Summaries. The compact GRASP representation of a property graph $G$ is computed as follows. A grouping phase partitions it into subgroups, based on the connectivity of its most frequent edge labels. An evaluation phase collapses the vertices and inner-edges of each maximally label-connected subgrouping. Finally, the source/target merge phase further collapses the common start/end nodes of same labeled edges.

Formally Verified Graph Querying

Graph queries used in commercial implementations are subsumed by a logic programming language called Datalog, which extends the expressivity of conjunctive queries with recursion. It consists of a set of facts and of if-then-else rules, allowing to infer new facts from existing ones. A Datalog program $I$ is evaluated by instantiating the heads of each rule with substitutions $\nu$ that uniformly match its body atoms against a set of facts $T$. The minimal model MM($I$) corresponds to the fixed-point iteration of $T_\infty = T \cup \{\nu(\text{head}) \mid \text{head} \in I \land \nu(\text{body}) \subseteq T\}$.

Formally Verified Graph Querying

Graph queries used in commercial implementations are subsumed by a logic programming language called Datalog, which extends the expressivity of conjunctive queries with recursion. It consists of a set of facts and of if-then-else rules, allowing to infer new facts from existing ones. A Datalog program $I$ is evaluated by instantiating the heads of each rule with substitutions $\nu$ that uniformly match its body atoms against a set of facts $T$. The minimal model MM($I$) corresponds to the fixed-point iteration of $T_\infty = T \cup \{\nu(\text{head}) \mid \text{head} \in I \land \nu(\text{body}) \subseteq T\}$.

Graph Querying in Datalog

A graph $G$ can be seen as database instance by considering edge labels to be binary predicates. Datalog rules can then capture reachability through complex paths, as computed by pfriends below. These can represent unions, conjunctions, compositions, and inverses of paths with labels belonging to a regular language. Querying for pair-wise label constrained reachability thus amounts to evaluating the result $V[G]$ of running a Datalog program $I$ over the instance $G$.

In practice, however, graph data is constantly evolving and prone to updates $\Delta$, i.e., to edge additions and removals. To avoid the costly recomputation of query results when a graph instance $G$ is thus modified into $G + \Delta$, one can instead maintain previously computed results. This amounts to incrementally updating the existing result, in order to produce the same answer as a full recomputation (see above diagram). The correctness of the algorithm amounts to proving that: if $V[G] \subseteq H$, then the engine outputs an incremental update $V^\Delta$, such that $V[G] + V^\Delta = H + \Delta$. We have used the Coq proof assistant to model the specification of a graph database that builds on the edge-labeled model and have implemented and formally proved an inference engine capable of both incremental evaluation and maintenance of graph queries belonging to the regular Datalog fragment.

Formally Verified Graph Querying

Graph queries used in commercial implementations are subsumed by a logic programming language called Datalog, which extends the expressivity of conjunctive queries with recursion. It consists of a set of facts and of if-then-else rules, allowing to infer new facts from existing ones. A Datalog program $I$ is evaluated by instantiating the heads of each rule with substitutions $\nu$ that uniformly match its body atoms against a set of facts $T$. The minimal model MM($I$) corresponds to the fixed-point iteration of $T_\infty = T \cup \{\nu(\text{head}) \mid \text{head} \in I \land \nu(\text{body}) \subseteq T\}$.

Graph Summary and Approximate Query Processing

Graph Summary and Approximate Query Processing

GRASP Summaries. The compact GRASP representation of a property graph $G$ is computed as follows. A grouping phase partitions it into subgroups, based on the connectivity of its most frequent edge labels. An evaluation phase collapses the vertices and inner-edges of each maximally label-connected subgrouping. Finally, the source/target merge phase further collapses the common start/end nodes of same labeled edges.

GRASP System


References