Query, Analysis, and Benchmarking Techniques for Evolving Property Graphs of Software Systems

Ph.D. Booklet

Gábor Szárnyas

Thesis supervisor:
Dániel Varró, D.Sc.

Budapest
2019
1 Introduction

1.1 A Brief History of Graph Data Processing

Although graphs have a history spanning to almost three centuries, they only became a primary tool of scientists in the second half of the 20th century. Starting from the 1960s, researchers studied a number of theoretical and practical aspects of graph processing and analysis. As a result, they established new fields by the 1970s, including graph rewriting systems \cite{EPS73} and social network analysis \cite{Gra73}. During this time, the database management research community defined the graph-based network data model, published by the CODASYL (Conference on Data Systems Languages) Consortium \cite{Dat70}. However, it had little influence as it was soon superseded by the less implementation-oriented and more flexible relational data model \cite{Cod70} that allowed system developers to decouple the query compiler from the query execution code and paved the way for using high-level declarative query languages.

As relational database management systems (RDBMSs) were both novel and dominant at the same time, there was little activity in graph data management for a few years. However, starting from the mid-1980s, researchers started to design and implement a number of object-oriented database management systems (OODBMSs). The 1990s saw a large number of theoretical works, research prototypes and commercial products in this space \cite{AG08}, which also exerted influence on RDBMSs, resulting in the implementation of object-relational systems. In the late 1990s, the semantic web initiative gave birth to the RDF (Resource Description Format) standard, with semantic databases following suit in the early 2000s. Around this time, researchers independently discovered that networks of both the World Wide Web \cite{AJB99} and the underlying internet topology \cite{FFF99} exhibit power-law distributions. These findings kickstarted the field of network science \cite{BP16}, which incorporates many of the previous results (such as those of social network analysis), but almost exclusively focuses on simple (directed or undirected) graphs.

Interestingly, none of the data models above achieved such a success in such a short amount of time as the property graph \cite{HG16, Ang+18, Fra+18, Ang18}, which extends directed graphs by only adding labels/types and properties (key-value pairs) to their nodes/edges. The first property graph database system appeared in 2007, just two years before the term “NoSQL database” \cite{SF12} was coined. While there are still significantly fewer property graph databases than other types of NoSQL systems (document databases, key-value stores, and wide column stores) \cite{DCL18}, graphs are increasingly used both for storing and processing highly interconnected data sets, and property graph databases have a dynamically growing market share\footnote{According to the statistics of the DB-Engines ranking site, the interest for graph databases between 2013 and 2019 has grown by 9.5 times (see \url{https://db-engines.com/en/ranking_categories}).}

We believe that the success of the property graph data model can be attributed to the fact that it strikes a good balance between conceptual simplicity and expressiveness. As the human mind tends to interpret the world in terms of things (nodes) and their respective relationships to one another (edges) \cite{Rod08}, along with attributes (properties) of these elements, property graphs provide a modelling framework that is intuitive but lightweight at the same time. Property graph instances are also better suited for visualization than other data models, such as RDF and hypergraphs \cite{LP91}. Still, other graph data models, such as the edge-typed graphs are still relevant. Therefore, this dissertation discusses techniques and applications both for graph databases \cite{Ang12, RWE15, Ang+17} storing property graphs and for analytical frameworks \cite{Bat+15, Sak+16, Yan+17} working on typed graphs.
1.2 Model-Driven Engineering

Model-driven engineering (MDE) is a widely used development technique in many application domains such as automotive, avionics, and other cyber-physical systems [WHR14]. MDE facilitates the use of models in different phases of design and on various levels of abstraction. These models enable the automated synthesis of certain design artifacts (such as source code, configuration files, documentation) and help catch design flaws early by model validation techniques. MDE originates from the fields of formal methods and software engineering (strongly influenced by object-oriented programming [Rum+91]), but established itself as a field of its own right in the last two decades. Many of the systems designed using MDE techniques have a long lifespan compared to most software products, e.g. a successful airliner is often produced and maintained for decades. Therefore, toolchains that build on open-source software (at least to some extent) are preferred to avoid vendor lock-in and aid the sustainability of the development tools.

In typical MDE tools, domain-specific models are defined as typed, attributed graphs. Therefore, processing and storing them requires techniques suited to efficient graph persistence and querying. A key component of MDE is model validation, i.e. checking whether a set of well-formedness constraints are satisfied on a domain-specific instance model. While RDBMSs also offer data validation in the form of integrity constraints or check constraints [RG98], these define simpler rules that enforce ranges or check for the presence of foreign key–primary key pairs. Model validation queries, on the other hand, often use complex graph queries and traversals for defining constraints, which are difficult to evaluate efficiently. Other workloads such as model simulation and defining view points also rely on complex queries, often executed repeatedly.

As the front-end components of MDE software applications are mostly programmed in statically typed object-oriented languages (such as Java), they require the underlying storage and query layers to provide strong metamodelling features, including defining and enforcing a graph schema with a number of constraints such as containment hierarchy, cardinalities, etc. This is in stark contrast with graph and semantic databases, which offer a schema-free or schema-optional data model [BTL11], and provide only basic metamodelling features. In short, MDE applications demand the following features:

- **Metamodelling.** They require strong metamodelling facilities beyond the capabilities of currently available graph and semantic database systems, which are schema-optional at best.
- **Complex queries.** They use complex and global queries for workloads such as model validation or model simulation, consisting of numerous join operations and accessing a large fragment of the graph model. Some queries also rely on features outside relational algebra such as transitive reachability and determining strongly connected components.
- **Frequent reexecution.** Queries are repeatedly evaluated over an evolving graph, including the addition of new graph elements, updating attributes, and deleting existing elements.
- **Low response time.** Many of the common MDE workloads define operations where execution time is constrained by usability requirements. Model validation, model simulation, and maintaining multiple view points all necessitate efficient, low response time (preferably sub-second) query evaluation.

In the last two decades, a number of dedicated graph and model transformation tools [Kah+18] were created to tackle these challenges. A possible approach to mitigate the difficulty of complex graph queries is using incremental view maintenance (IVM) techniques, which aim to keep query results up-to-date efficiently upon changes in the graph (depicted in Figure 1.1). These techniques have been used originally in production systems [For79] and active databases [PD99] that use triggers to maintain views [Sto75] in a consistent state. In the context of model-driven engineering, IVM techniques have been employed by the VIATRA tool [Var+16] (originally developed at the Fault-Tolerant
Systems Research Group of the Budapest University of Technology and Economics), along with Reactive ATL, eMoflon, and NMF. However, a number of other features such as scalability [Kol+13], fine-grained access control [Deb+17], and inconsistency tolerance [GHM98] would be desirable to provide a usable persistence and query backend for MDE applications. Due to the lack of off-the-shelf solutions providing these features, MDE applications commonly use custom model management layers that implement these features to some degree.

Figure 1.1. Incremental view maintenance at a glance, adapted from [CY12]. Query $Q$ is evaluated over graph $G$, returning $Q(G)$. Once changes $\Delta G$ are applied to the graph, it becomes $G + \Delta G$. The changes are subsequently reflected in the query results $Q(G + \Delta G)$, either through full recomputation or by applying $\Delta Q(\Delta G)$ to the previous query result.

1.3 Model Queries over Databases

At first sight, database systems seem a good fit for storing and processing the graph models of MDE applications. Due to their long history and sophisticated optimizations, databases are expected to be more scalable for complex queries than MDE query engines. Additionally, they offer better inconsistency tolerance, i.e. they allow persisting incomplete and therefore malformed graph instances unlike many MDE tools. They also support multiple transactions to run read/write operations concurrently. As they are often used in enterprise environments for large collaborative projects involving multiple stakeholders, they provide fine-grained access control, backup, and crash recovery features. The MDE community therefore been long experimenting with adapting database technologies to its applications [Var08].

Relational database management systems There have been multiple attempts to create mapping layers between MDE tools and RDBMSs, the two most prominent being the Eclipse Teneo [Ecl15] and Connected Data Objects (CDO) projects [Ecl19]. Still, relational databases have been repeatedly falling short in terms of performance and usability for supporting MDE workloads [BHH12; Góm+15; Sey+16; Hae+19]. The key reasons behind this are as follows:

- Using RDBMSs for storing graph models necessitates object-relational mapping (ORM) [Bla+06; ONe08] which suffers from the impedance mismatch problem [Ire+09].
- Graph traversal queries are difficult to translate to SQL [VFV06]. Even though support for recursive queries was introduced as an extension to the SQL:1999 standard, it was only adopted by MySQL, the most popular open-source RDBMS, in 2017. Even with this language construct, recursive queries are often cumbersome to express and optimizing their evaluation requires a great deal of expertise.
- Relational databases are notorious for having an excessive amount of configuration parameters to fine-tune performance [Ake+17]. While these can be used to improve performance for a given workload, getting them right for a certain workload requires a great deal of experience.
The popular open-source implementations (MySQL and PostgreSQL) do not support incremental query evaluation, and even the proprietary ones (Oracle Database or Microsoft SQL Server) support it only to a limited extent.

**Object-oriented database management systems** Following the rise in popularity of the object-oriented programming paradigm, *object-oriented database management systems* (OODBMSs or *object databases*), specializing in efficient retrieval of elements from object graphs, started to emerge in the 1980s. These systems were met with particular interest by the developers of *computer-aided design and manufacturing* (CAD and CAM) tools, which share many challenges of MDE workloads (such as using statically typed programming languages for front-end development), but aim at concrete application domains.

OODBMSs were initially successful, and by the mid-90s, more than 30 such systems were available [ZCC95], including commercial systems such as ObjectStore and O₂ [BDK92] along with research prototypes such as H-PCTE [Kel92] and GRAS [KSW95]. A new standardized query language, the *Object Query Language* (OQL) [CB00] was designed for OODBMSs, providing a syntax similar to SQL but expressing navigation operations in a functional nature. Despite the number of available systems, OODBMSs had limited success outside the narrow domains of CAD/CAM tools with only moderate adaptation to MDE and other tools. The reasons behind this are manifold, but we believe the key contributing factors were the complexity of the data model, poor performance (compared to RDBMSs) and the lack of open-source implementations. In short, neither RDBMSs nor OODBMSs turned out to be suitable backends for storing graph models. Therefore, following the appearance of the non-relational NoSQL and semantic databases, MDE researchers soon started to investigate them for storing and manipulating graphs.

**NoSQL and semantic databases** Recognizing that the challenges in MDE are often closer to those of object-oriented, semantic, and NoSQL databases, the software engineering and MDE community has been experimenting for a long time to adapt database technologies to store object models [Rah+01; VFV06]. In recent years, they investigated the applicability of NoSQL systems for tackling scalability challenges [Kol+13], including persistence layers (EMFStore, Morsa, NeoEMF), query evaluation (Mogwai), and model indexing (Hawk). The *Open Services for Lifecycle Collaboration* (OSLC) [BGL12], an open community started in 2008, aimed to harness semantic technologies to assist software engineering efforts. To this end, it provides a common approach for tool integration that build on top of the RDF standard [RS14], the Linked Data method [BHB09], and the REST (Representational State Transfer) software architecture style [BB08]. While many of these efforts were successful to some extent, the overall results and their adoption suggests that NoSQL systems are not yet fully capable of supporting complex MDE workloads, particularly struggling to provide sufficient performance for the graph queries required by such workloads.

**The complexity of MDE workloads** In short, to sufficiently handle complex MDE workloads, users require continuously updatable graph database that supports strong metamodelling, can be queried with an expressive declarative query language, and can evaluate complex global queries with close to real-time response. At the beginning of my research in 2014, such systems did not yet exists—even the concept of a “graph data warehouse” has only been explored in research works [Zha+11; LV13], and as of 2019, such systems are still not available yet.
1.4 Benchmarks for Describing Application Scenarios

To study the challenges of handling complex graph query workloads and conduct research with impact on industry tools, we need to consider real-life use cases. This itself is a challenge as it is impossible to obtain (let alone publish) real workloads in model-driven engineering and graph data processing. The reason behind this is that both real data sets and even queries specifications constitute considerable intellectual property. Data sets might hold sensitive personal information (such as financial and medical data) or accidentally identify subcontractors (e.g. from the structure of an automotive component’s model). In many cases, the queries are also difficult to obtain, especially in cases when exposing them might be considered a risk to the business (e.g. the queries used for financial fraud detection can give hints on how to circumvent such anti-fraud measures).

The best practice to address these challenges is to rely on commonly accepted benchmark specifications. Such benchmarks have a number of additional advantages, but most importantly, they make competing products and approaches comparable, thus stimulate research and accelerate technical progress in their field [Pat12], as the Transaction Processing Performance Council’s TPC benchmarks [TPC10; TPC18] have done for the RDBMS industry during the past 30+ years. While there are a number of benchmarks available for graph workloads [VSV05; Sch+09; Arm+13], none of them provides comparative measurements for complex graph queries on realistic data. At the beginning of my research in 2014, benchmarks targeting such workloads – the Train Benchmark and the LDBC SNB’s Business Intelligence workload – were in an early stage of research. This dissertation makes major contributions to both of these benchmarks.

Designing representative workloads  On the conceptual level, a benchmark specification defines (1) data sets, e.g. graph instances of increasing sizes, (2) a set of queries, and (3) a set of scenarios that prescribe the sequence of operations to be performed. A key requirement for benchmarks is that they should be representative [Gra93], i.e. their data sets, queries, and scenarios should resemble real use cases that are relevant to the interest of benchmark users. In this dissertation, we aim to assist benchmark designers in this goal by proposing a set of abstract characteristics for characterizing a given workload. As discussed previously, users often cannot share the queries used in their workloads. However, they might be able to answer questions such as “What is the maximum number of joins used in a query?”, “What percentage of the database is accessed by a query?” and “Which types of aggregation operations are the bottleneck?”. These inputs can be used to construct similar queries.

It would be logical to apply the same approach for creating representative data sets. However, existing graph benchmarks often use random graph models [Tae+07] that exhibit a highly regular structure or graphs that correspond to a relational data set [BS09]. Even the most recent and advanced approaches such as gMark [Bag+17] only consider a narrow set of characteristics (e.g. degree distributions can be controlled for a given edge type, but the interplay between types cannot be tuned). While there would definitely be interest for realistic graph generators [SLOT18], both characterizing real graphs and synthesizing realistic graphs are highly non-trivial problems. In this dissertation, we adapt recent results of the multidisciplinary field of network science to obtain metrics that describe the structure of typed graphs. Synthesizing realistic graphs is an open research challenge and state-of-the-art approaches offer limited scalability, only supporting graphs up to a few hundred elements [You+18].
2 Challenges

As we concluded in Section 1.3, MDE tools could greatly benefit from using high-performance graph database systems, especially ones supporting incremental view maintenance over property graphs or semantic graphs. Other users also expressed their interest for running continuous queries over an evolving graph data set. While such systems were already proposed in research works in the 1990s [KSW95], no graph database system offered incremental query evaluation at the beginning of my research in 2014. This still holds true as shown by recent survey [Sah+17], which interviewed users from industry and academia about their graph data management interest and practices. In fact, this survey confirmed both the demand for incremental graph processing techniques (more than a third of respondents indicated that they perform some incremental computations on their graphs) and the lack of such systems (the 22 software products included in the survey have limited or no support for incremental computations), suggesting that users who indicated that they rely on IVM use either computations with limited incrementality and/or implemented problem-specific ad-hoc solutions.

To create the building blocks of an incremental graph query engine that work both in the MDE and property graph domains, we investigated the challenges of scalable incremental view maintenance for graph queries. To derive representative performance results for such systems, we looked for representative macrobenchmarks to measure the performance of global graph queries. Finally, we aimed to characterize realistic graph models.

Ch1. Queries over evolving property graphs. A common approach to speed up the evaluation of queries on continuously changing data sets is incremental view maintenance (IVM), which defines a view for each query and maintains its results upon changes. While IVM techniques have been developed for more than three decades, the feasibility of incremental operations on graph data structures is still actively studied from the theoretical perspective [FHT17]. Additionally, the sophisticated property graph data model introduces even more challenges for IVM techniques.

Ch2. Representative macrobenchmarks for graph querying. To capture challenging aspects of real workloads, we need some abstract aggregative descriptions to provide a summary of why they are difficult. This characterization is important for two key reasons: (1) it allows benchmark designers to define synthetic but representative workloads without requiring access to confidential information such as queries and data sets (both protected by intellectual property rights and non-disclosure agreements) and (2) it makes possible to combine different use cases into a single one workload. These abstract descriptions include ones that target the data set, such as typed graph metrics, and also ones that define query language features or stress performance aspects.

Ch3. Domain-specific characterization of realistic graph models. Creating realistic graph instances for benchmarks and distinguishing synthetic graphs from real ones requires an approach to characterize graph models in a certain domain. The field of network science has studied a number of graph metrics (such as degree distribution) to characterize real networks, however, devoted less attention to metrics that characterize graphs with type information, which is necessary to describe domain-specific graph models.

Figure 2.1 lays out the challenges and connects them to the proposed contributions. It shows that the central themes of this dissertation are global graph queries, macrobenchmarks, and realistic workloads, with benchmarks closely related to each of the three contribution groups.
3 Contributions

3.1 Structural Analysis of Typed Graphs

How does a network emerge and what patterns does its graph exhibit? These are central questions of network science that have been studied to great depth during the last few decades. However, these works almost exclusively studied homogeneous networks, i.e. networks of a single node label and a single edge type, which can be modelled with an untyped graph. While this approach yielded groundbreaking results in many areas (such as the analysis of biological and social networks), it is difficult to apply in fields which use heterogeneous graphs with edge types (also known as multiplex networks). Types introduce even more variety in the structural interplay between the nodes and edges of the graph. On a fine-grained level, this could mean investigating how likely it is that two friends of a given person engage in common activities. At a higher granularity, it would be interesting to observe how communities are formed based on such heterogeneous triangles. These examples show how in-depth structural analyses that take the types into account could lead to a better understanding of the emergence of both the microscopic and the macroscopic structure of typed graphs.

Generating synthetic graph instances is an important research challenge that has a variety of use cases, ranging from producing customer data for benchmarking to creating test graphs for self-driving autonomous vehicles. To implement such graph generators, we first have to formulate a basic understanding of how the target graphs are structured. This allows us to guide the generator so that the synthesized graph satisfies the desired structural properties.

Group of contributions 1 I proposed various graph metrics and statistical analysis techniques to characterize domain-specific engineering models and graph generators.

1.1 Multidisciplinary graph metrics. I adapted graph metrics originally proposed in network science to describe typed graphs of systems engineering models. [c4]

1.2 Characterization of engineering models. I proposed characteristic graph metrics to distinguish graph models from different domains using statistical analysis. [c4]

1.3 Characterization of synthetic graphs. I identified graph metrics to characterize synthetic graphs derived by various graph generators in order to distinguish them from real graphs. [e1]
**Added value**  In this work, I proposed a domain-agnostic extendable method to characterize graph models, which can be applied on any graph model. The graph metrics collected during this research allow users to estimate the graph data in real application scenarios by providing a toolkit that characterizes the complex structure of graphs without revealing detailed and potentially sensitive information, thus respecting privacy and intellectual property rights (IPR). Finally, I show that these metrics can be used to differentiate between real and synthetically generated graph instances, thereby identifying and quantitatively characterizing the limitations of existing state-of-the-art graph generation approaches.

**Previous and related results in the research group**  Members of our research group investigated related graph and query metrics in [Izs+13b] to characterize and predict performance of graph queries to assess how much worse a query engine performs compared to a theoretical lower bound. A key difference of my work is to characterize domain-specific graph models and model generators (and not the queries themselves), which is an orthogonal use of graph metrics. Related graph metrics have been evaluated for generic graph models in order to study their effect on query performance by Zsolt Kővári in [Kőv15], which was co-supervised by István Ráth and myself. Zsolt Kővári and Ágnes Salánki contributed the statistical analysis of graph models in our joint paper [c4]. Oszkár Semeráth implemented the model generator to produce synthetic statechart models used in our joint paper [e1].

**Open-source software**  The framework for analyzing typed graphs is available as an open-source project[^1]. The project consists of approx. 5 000 lines of Java code for loading the graphs and calculating graph metrics, along with 500 lines of R code for visualization and data analysis. Additionally, a less scalable but more visual demo implementation, using the Neo4j graph database [Web12] and its Cypher query language [Fra+18], is available online[^2].

### 3.2 Benchmarks for Global Queries over Evolving Property Graphs

*Standard benchmarks* specify a workload and inspect the behaviour of various implementations executing it, measuring a set of metrics (response time, energy consumption, correctness, etc.). It is established that benchmarks have the power to shape a field [Pat12] and accelerate its progress, particularly for new and emerging areas, which are yet to establish a common understanding of what the focal points of the field should be. In such cases, a de-facto benchmark accepted by members of the community allows competing tools to be compared using a standard and precisely specified workload. Benchmarks which sufficiently cover important features also free authors from the burden of designing and implementing their own ad-hoc benchmarks (which might be biased towards their tools), and aid the reproducibility of their performance experiments.

**Group of contributions**  I contributed to the design of two benchmark frameworks for global property graph queries. My specific contributions include categorization of query language features, numerous enhancements to the benchmarks as well as experimental evaluation.

2.1 **Expressivity requirements for query languages**. I identified abstract and qualitative language features (choke points) to systematically assess the expressivity requirements for graph queries used in performance benchmarks. [e17]

[^1]: https://github.com/ftsrg/graph-analyzer
2.2 Adaptation of benchmarks for the property graph data model. I adapted two open benchmarks, the Train Benchmark and the LDBC Social Network Benchmark, for property graphs, including the implementation of scalable data generators, novel global queries, and deterministic update transformations. \[1, e17\]

2.3 Extensive experiments with benchmarks. I conducted an experimental evaluation and exploratory analysis of different query technologies over static RDF and evolving property graphs under various workloads. \[1, c7, e17\]

**Added value** Up to our best knowledge, both benchmarks target a unique workload. The Train Benchmark is first cross-technology benchmark that measures the performance of repeated model validation operations, considering representation formats, query languages and query engines from multiple technological spaces (EMF, RDF, property graph, and SQL). The LDBC SNB’s Business Intelligence workload is the first benchmark for decision support-style, aggregation-dominated global graph queries, which, through systematic choke point-driven design process, covers challenging features both language- and performance-wise.

**Previous and related results in the research group** Benchmarking of graph-based query and transformation tools has been in the focus of the research group and investigated in a series of papers such as [SV05, Hor+10, Ujh+15]. The first version of the Train Benchmark was proposed by Benedek Izsó, István Ráth, Balázs Polgár, along with Zoltán Szatmári whose dissertation contained contributions to the initial version of the benchmark [Sza17]. Since 2012, I contributed to all major components of the benchmark in close collaboration with many colleagues (also including Oszkár Semeráth and Gábor Bergmann). My contribution statements presented above exclude inseparable joint contributions and claim results in the context of property graphs where my own work was predominant. In case of the LDBC Social Network Benchmark, I extended existing update operations to include removal of elements, which is a major challenge for incremental query evaluation engines, and also resolved a number of ambiguities in the benchmark specification.

**Open-source software** The Train Benchmark is available as a single open-source project [https://github.com/ftsrg/trainbenchmark][3]. It contains approx. 18 000 lines of Java code, 800 lines of Groovy code and 600 lines of R code. The LDBC Social Network Benchmark consists of multiple projects with contributions from dozens of authors. Most of my work manifested in the specification [https://github.com/ldbc/ldbc_snb_docs][4], the workload driver [https://github.com/ldbc/ldbc_snb_driver][5] and the reference implementations project [https://github.com/ldbc/ldbc_snb_implementations/][6] where I performed extensive refactoring and added a number of new query implementations. My contributions in the latter account for approx. 4 000 lines of Java code and 800 lines of Cypher code. I also performed various maintenance tasks in the data generator [https://github.com/ldbc/ldbc_snb_datagen][7].

3.3 Incremental View Maintenance on Schema-Optional Property Graphs

Efficient query evaluation – incorporating many techniques from indexing through query optimization to join processing algorithms – is one of the holy grails of database research. In many workloads,
especially in OLAP-style data processing, queries are known in advance and are evaluated against a continuously changing data set. In these cases, query processing can be sped up by defining materialized views and using incremental view maintenance (IVM) to keep their content in sync with the changes in the data. Today, IVM is supported by many (mostly commercial) relational database systems, and would certainly be of interest in the graph processing space. However, while IVM has more than three decades of literature, most works only considered the relational data model, and only a few proposed IVM algorithms for the nested and graphs data models. Additionally, the schema-optimal approach of property graph databases and the bottleneck of single node systems make the application of IVM even more challenging.

**Group of contributions**

I developed scalable incremental view maintenance techniques for evolving schema-optional property graphs.

3.1 **Schema inferencing from property graph queries.** I designed a schema inferencing algorithm from nested to flat relational algebra that deduces the relevant minimal schema of the property graph from query specifications. [e16; r20]

3.2 **Mapping from property graph queries to nested relational algebra.** I defined a chain of consecutive mappings from high-level property graph query languages to nested and flat relational algebra to enable the use of traditional relational query evaluation and optimization techniques. [c5; r20]

3.3 **Parallel query processing for incremental view maintenance.** I proposed an asynchronous graph query technique for incremental view maintenance that uses Rete-based query evaluation of flat relational algebra over schema-optional property graphs. [e16; r20]

3.4 **Scalable distributed query evaluation.** I proposed an asynchronous execution strategy and a termination protocol by combining the actor model with Rete-based query evaluation for scalable incremental view maintenance on RDF graphs. [e8; c3; e13]

3.5 **Experimental evaluation of incremental graph query processing.** I evaluated the performance and scalability of a prototype implementation of for parallel execution (ingraph) using the LDBC Social Network Benchmark. I adapted the Train Benchmark for a cloud-based execution environment to carry out scalability evaluation of a prototype implementation for the distributed approach (IncQuery-D). [c3; r20]

**Added value**
The compilation and transformations steps presented in this thesis can be used to reduce a large set of property graph queries to flat relational algebra and thus allows query engine developers to apply existing IVM techniques for these queries. As IVM incurs a significant memory overhead, I also studied the possibility of using a distributed setup to improve the scalability of the approach for large graphs. I believe one of the most future-proof contributions of this thesis is the identification of the required challenges to adapt IVM techniques for property graph queries. While some of these challenges were already addressed in this dissertation, and some can be tackled by adapting existing solutions, many of them are open research problems.

**Previous and related results in the research group**
Gergely Varró designed efficient search-based graph query algorithms on graph models [Var08] which was further extended by Ákos Horváth for hybrid search plans [Hor+10] and to EMF models by the development team of the eMoflon and VIATRA tools in [Var+15; Búr+15]. Gábor Bergmann’s work discussed incremental view maintenance on EMF models [Ber13] and proposed an initial version for a custom parallelization of Rete network, which is generalized in my work to a distributed actor-based environment. István Ráth presented
event-driven and incremental model transformation techniques in his dissertation [Rát11]. Furthermore, he was the main contributor of the IncQUERY-D architecture. József Makai contributed allocation and reconfiguration strategies for IncQUERY-D [e13]. József Marton defined a formalization of the openCypher language [c5]. János Maginecz made major contributions to the ingraph prototype, including various optimization techniques and significant implementation efforts [e14; j2; r20]. Csaba Debreceni proposed novel incremental synchronization for secure view models to support collaborative development [Deb19].

Open-source software IncQUERY-D [e8; c3; e13; e14] was implemented in Java and Scala, using the compiler of VIATRA Query. Its distributed and sharded extension, IncQUERY-DS (implemented purely in Scala) is available as a separate project [e14]. ingraph [j2; c3; c6; e16; r20] is a mix of Scala and Java code, where I contributed to all components, including the compiler, the data loader and the incremental query engine.18

18https://github.com/viatra/incqueryd
19https://github.com/viatra/incquery-ds
20https://github.com/ftsrg/ingraph
Related software projects The IncQuery Server for Teamwork Cloud \cite{Heg+18} is a server-side query middleware service for collaborative modelling. Like IncQuery-D, it supports distributed processing, but on a different (microservice) level of granularity. Furthermore, it is also based on the reactive programming paradigm, but uses the Eclipse Vert.x library\footnote{https://vertx.io/} instead of Akka actors used in IncQuery-D.

Summary of Contributions

Figure 3.1 shows an overview of the groups of contributions, including their key results and prototype tools, positioning them w.r.t. data processing system types (such as model query engines and relational databases), industry tools, and workload categories.

4 Open Challenges and Future Work

We believe that the work presented in this dissertation will open up a number of interesting research directions. Here, we highlight key directions for future research.

Analysis of graphs using meta-paths We plan to adapt state-of-the-art graph analysis approaches such as meta paths \cite{Shi+17}, and to apply proven metrics such as the \textit{k-local clustering coefficient} to typed graphs \cite{JC04,Fro+02}. To improve the scalability of analysis workloads, we already reformulated multiple challenging metrics in the language of linear algebra \cite{Var18} and plan to experiment with them on large data sets. We are also looking into other means to improve scalability, including recent advancements in elastic graph processing \cite{Uta+18}.

Graph synthesis We plan to combine the \textit{characteristic graph metrics} identified in the dissertation with the work on diverse and consistent graph generation by Oszkár Semeráth \cite{Sem19} to \textit{synthesize} consistent, realistic, diverse, and scalable graph models.
Publications

Publication List

<table>
<thead>
<tr>
<th>Number of publications:</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of peer-reviewed journal papers (written in English):</td>
<td>2</td>
</tr>
<tr>
<td>Number of articles in journals indexed by WoS or Scopus:</td>
<td>1</td>
</tr>
<tr>
<td>Number of publications (in English) with at least 50% contribution of the author:</td>
<td>4</td>
</tr>
<tr>
<td>Number of peer-reviewed publications:</td>
<td>22</td>
</tr>
<tr>
<td>Number of independent citations:</td>
<td>58</td>
</tr>
</tbody>
</table>

Publications Linked to the Contributions

<table>
<thead>
<tr>
<th>Thesis 1</th>
<th>Thesis 2</th>
<th>Thesis 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>[c4]</td>
<td>[j1]</td>
</tr>
<tr>
<td>—</td>
<td>[c7]</td>
<td>[c3]</td>
</tr>
<tr>
<td>—</td>
<td>—</td>
<td>[c5], [c6], [e8], [e10], [e13], [e14], [e16]</td>
</tr>
<tr>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

This classification follows the faculty’s Ph.D. publication score system.

Book Chapters


Journal Papers


International Conference Papers


Workshop, Symposium, and Contest Papers


Technical Reports


Other Publications (Not Linked to Contributions)

Workshop, Symposium, and Contest Papers


[Sak+16] Sherif Sakr, Faisal Moeen Orakzai, Ibrahim Abdelaziz, and Zuhair Khayyat. *Large-Scale Graph Processing Using Apache Giraph*. Springer, 2016. doi: [10.1007/978-3-319-47431-1](https://doi.org/10.1007/978-3-319-47431-1).


[Zha+11] Peixiang Zhao, Xiaolei Li, Dong Xin, and Jiawei Han. Graph cube: On warehousing and OLAP multidimensional networks. In: *SIGMOD*, pp. 853–864. ACM, 2011. doi: [10.1145/1989323.1989413]