AvantGraph Query Processing Engine

Wilco v. Leeuwen, TU Eindhoven, Eindhoven, NL, w.j.v.leeuwen@tue.nl
Thomas Mulder, TU Eindhoven, Eindhoven, NL, t.mulder@tue.nl
Bram van de Wall, TU Eindhoven, Eindhoven, NL, a.a.g.v.d.wall@tue.nl
George Fletcher, TU Eindhoven, Eindhoven, NL, g.h.l.fletcher@tue.nl
Nikolay Yakovets, TU Eindhoven, Eindhoven, NL, hush@tue.nl

ABSTRACT

We demonstrate AvantGraph, a graph query processing engine developed by the Database group at TU Eindhoven. Designed for efficient processing of *both* subgraph matching and navigational graph queries, AvantGraph encompasses innovation in three key areas: the planner, the cardinality estimator, and the execution engine. We present demonstration scenarios covering a wide range of workloads across diverse domains which (1) provides deep insights into the core challenges of complex graph query processing and (2) showcases corresponding critical optimizations via "under-the-hood" operational insights of AvantGraph's key components.

PVLDB Reference Format:
Wilco v. Leeuwen, Thomas Mulder, Bram van de Wall, George Fletcher, and Nikolay Yakovets. AvantGraph Query Processing Engine . PVLDB, 15(12): 3698 - 3701, 2022.
doi:10.14778/3554821.3554878

1 INTRODUCTION

Graph databases have enjoyed increased attention in recent years both in the academic and practitioner communities. Representing information as a graph is appealing due to the simplicity and familiarity offered by graph-based data models. For example, the *property graph* data model (PGM [2]), allows for a natural encoding of the data *topology* via graph elements such as vertices and edges while non-topological information is encoded in property *tables* attached to these graph elements. In addition to convenience of representation, graph databases facilitate complex analytical tasks that involve advanced navigation over stored networks. To achieve this, recent versions of popular graph query languages such as Cypher[1], SPARQL[2] and the upcoming GQL standard[3] all include syntactic constructs enabling graph navigation via variants of regular path queries (RPQs [10]), their conjunctions (CRPQs) and unions (UCRPQs). While *non-navigational* graph querying tasks can be typically solved by efficient algorithms for the subgraph matching (SGM) problem and its variations, complex graph navigation requires fundamentally different approaches and solutions. Indeed, SGM translates well into traditional relational-style processing routines leveraging the vast amount of expertise on optimization of queries over relational databases (e.g., SQL query optimization). On the contrary, graph navigation (e.g., with UCRPQs) has been studied significantly less and requires optimization over *recursive* or *iterative* fragments of the corresponding query algebras.

Existing solutions for recursive graph navigation either rely on trivial extensions of relational algebras (e.g., using variations of a transitive closure operator [9, 13]) or use intricate ad-hoc constructs (e.g., variations of finite automata [20]) which are difficult to embed and use in deeply optimized (e.g., vectorized, compiled, factorized, and worst-case optimal) processing pipelines. The former solutions miss optimization opportunities due to the limited plan space they can afford and the latter approaches miss out on the vast expertise on the optimization of execution of tuple-based data processing.

To bridge this gap, we developed AvantGraph[4], a next-generation graph query processing system which *raison d'être* is to process queries which contain *both* subgraph-matching and navigational fragments in a *single* cross-optimized pipeline. To achieve this, we make contributions in three key areas: (1) to enable efficient recursive processing, we propose Magellan - a top-down query planner that enumerates through and emits a new rich class of *cyclic graph* execution plans; (2) to process these novel plans, we introduce QuickSilver, a multi-threaded cyclic-plan execution engine which operates over factorized intermediate results (IR), uses worst-case optimal joins when advantageous, and is optimized with vectorization and query/predicate compilation; (3) finally, to aid our new planner, we propose BallPark - a cardinality estimation framework which enables the systematic principled mix-and-matching of the state-of-the art cardinality estimation techniques with the aim of producing superior estimates for diverse property graph queries.

We further developed TunEx, a tune-and-explain toolkit to provide deep insight into the core challenges addressed and the operating characteristics of the key components of the system. We use our toolkit to build a comprehensive demonstration scenario over both synthetic and real-world workloads from various diverse application domains.

2 GUIDED TOUR OF THE SYSTEM

AvantGraph is designed to be a high-performance, minimum-dependency processing engine for analytical queries over property graphs. The engine is implemented in modern C++ and employs low-level optimizations that reduce performance degradation due to lack of locality, branch mispredictions and non-uniform memory access. AvantGraph is a *polyglot* engine supporting inputs in both PGM and RDF[5] data models. For queries, feature-subsets of Cypher and SPARQL query languages are supported (see Fig. 1 for system overview).

Proceedings of the VLDB Endowment, Vol. 15, No. 12 ISSN 2150-8097.
doi:10.14778/3554821.3554878

[1]http://docs.neo4j.org/chunked/stable/cypher-query-lang.html
[2]https://www.w3.org/TR/sparql11-overview/
[3]https://www.gqlstandards.org
[4]http://avantgraph.io
[5]https://www.w3.org/RDF/

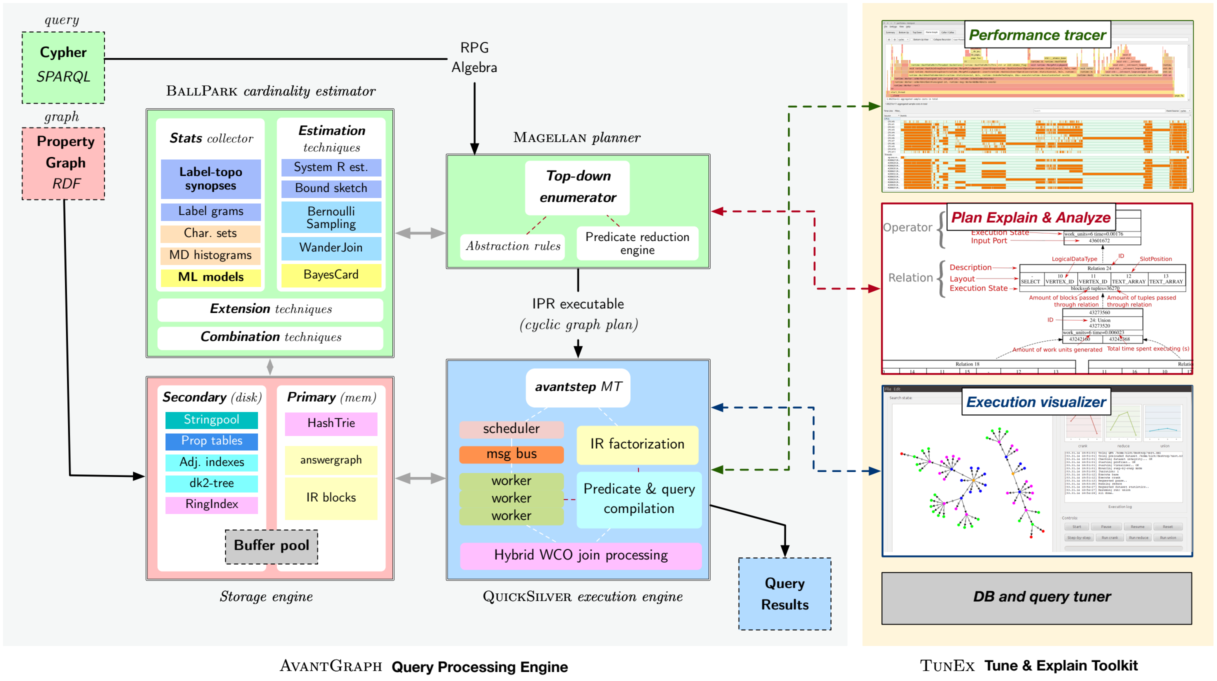

Figure 1: Overview of the AvantGraph query processing engine and its performance analysis and tuning toolset

During processing, user queries are translated into an internal representation based on the regular property graph algebra (RPGA [5]). RPGA is an algebra equivalent in expressiveness to regular property graph logic (RPGLog), an extended fragment of standard non-recursive Datalog designed to work with the PGM data model. RPGLog, at its base, considers *graph predicates* which can be unary (node labels) or binary (edge predicates). Further, RPGLog supports queries which operate over unions of conjunctions of graph predicates, augmented with edge variables and extended with a transitive closure operation and query nesting. The desirable characteristics of RPGA/RPGLog are two-fold. First, current practical query languages (e.g., Cypher and SPARQL) can be expressed in RPGA. In fact, RPGA is strictly more expressive than the corresponding fragments in both Cypher and SPARQL and, hence, is "future-proof" as it supports features that form the core of the upcoming languages such as G-CORE [3] and GQL (e.g., full support for UCRPQs and composability). Second, while being more expressive than current practical query languages, RPGA does not have higher evaluation complexity (NP-complete in combined complexity and NLogspace-complete in data complexity [5]).

AvantGraph's storage engine follows classical primary (memory) and secondary (disk) storage pool separation. To offset the negative effects of the secondary storage IO cost, we employ an efficient and scalable buffer manager which shares many properties with the recently proposed high-performance Umbra [12] relational query processing engine. Further, in addition to traditional adjacency indexes, string pools, and property table storage, AvantGraph includes native implementations for dk^2-trees [7] (for compressing adjacency), and RingIndex [4] and HashTrie [8] (for worst-case optimal processing).

2.1 Planning Recursive Analytics

Navigational queries over graphs (e.g., UCRPQs) require constructs which enable recursive or iterative processing in the corresponding query algebra. Most database techniques approach recursive processing from one of two perspectives: some extend the internal relational algebra (RA) with a variation of a transitive closure (TC) operator, while others use ad-hoc constructs based on specialized mechanisms derived from finite automata (FA). Algebra-based approaches are efficient, but miss plans [20], and automata-based approaches have rich plan spaces but are oblivious to the optimizations over the non-navigational fragments of a query.

AvantGraph borrows the best from both approaches by introducing Magellan: a planner that enumerates plans which are *cyclic* (cf. tree- or DAG-shaped plans used in other query processing engines) and operate in a single cross-optimized navigational and subgraph-matching execution pipeline. While cyclic plans allow us to encode any FA-plan (any automaton can be translated directly to a cyclic graph plan) and any RA-plan (iteration in the TC operators is represented by simple loops in a cyclic plan), they come at increased enumeration complexity as standard tree-algebra-based dynamic programming enumerators no longer work.

Instead, the enumerator works in a *top-down* manner and operates on *abstractions* over algebraic expressions which are typically, but not necessarily, sub-expressions of the input query. Any enumeration procedure starts from a single abstraction over the input query. Abstractions are represented schematically using a box containing the abstracted expression, and are iteratively *refined* until no abstractions are left. Refinement means replacing an abstraction by some concrete logical operator and zero or more new abstractions.

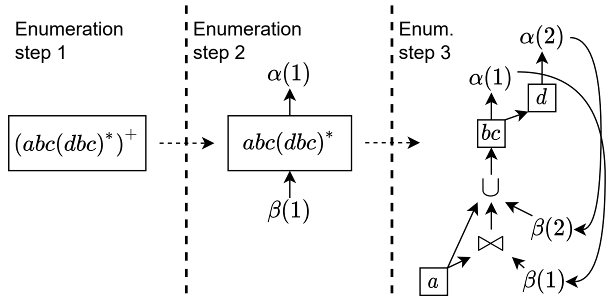

Figure 2: Example enumeration procedure. Solid arrows indicate tuple flow. Dashed arrows indicate enumeration steps. $\alpha(1)$ is the root operator.

To represent cyclic plans, two operators $\alpha(i)$ and $\beta(j)$ are introduced alongside conventional operators like join and union. The α and β operators are references to a named sub-plan. These references can be used recursively to construct a cyclic tuple flow. The informal semantics of an operator $\alpha(i)$ are to obtain all tuples from it's child operator, write them to a buffer i and pass them on to a parent operator. An operator $\beta(j)$ reads tuples written to a buffer j since $\beta(j)$ was last evoked, eliminates duplicates, and passes tuples on to a parent operator.

Figure 2 shows an example of one of the possible enumeration procedures for an input RPQ defined by $(abc(dbc)^*)^+$. Step 1 shows the abstraction over the input, step 2 establishes a cyclic tuple flow via the buffer 1, and step 3 constructs a tuple flow where the extension of a path with edges labelled b and c consecutively is *shared* between the two cyclic flows over buffers 1 and 2 which implement the Kleene plus and Kleene star, respectively.

Magellan implements all of the optimizations and considers a plan space which fully subsumes those of extended-RA approaches (α-RA [13], μ-RA [9]) and FA approaches (WaveGuide [19]). Further, note that Magellan's plans are strictly more expressive than μ-RA plans since, for example, the plan shown in Fig. 2 is not expressible in μ-RA.

2.2 Cardinality Estimation For PGM Queries

Cardinality estimation over property graphs add additional complexity related to the schemaless nature of stored graphs, many-to-many relationships encoded in the graph topology, and diverse (and often highly correlated) property predicates.

We address these by introducing a modular mix-and-match cardinality estimation framework for property graphs. Specifically, AvantGraph's cardinality estimator, BallPark [17], implements a collection of state-of-the-art estimation techniques based on graph synopses, sketches, histograms, machine learning models, and sampling. The idea is to address the complexity and diversity of graph workloads by systematically using, extending, and then combining the results of cardinality estimation techniques each of which are best suited for a query fragment at hand. For example, topological fragments of a query are best estimated with labeled topological synopses for a corresponding query shape (e.g., chains for navigational fragments, stars for relational-style subgraph matching). On the other hand, for highly-correlated property predicates, machine-learning-based approaches which perform inference over joint distributions are well suited.

2.3 Execution Engine For Cyclic Plans

AvantGraph's execution engine, QuickSilver, is designed for efficient and scalable processing of the novel cyclic physical execution plans produced by the Magellan planner. Towards this, we propose and implement novel optimizations in: (1) fine-grained intra-operator parallel execution, (2) use of factorized intermediate results, and (3) vectorized and compiled query runtime.

QuickSilver's parallel execution engine, AvantStep [16], is inspired by a recent QuickStep scale-up data platform [14]. The engine is optimized to have high intra-operator, intra-query, and inter-query parallelisms. Further, special care is taken to enable execution of cyclic plans, e.g., operators in a cycle are checked for completion *simultaneously* to prevent deadlocks. Specialized algorithms including worst-case optimal joins and transitive closure computation are also parallelized.

QuickSilver operates over both non-factorized (tuple blocks) and factorized IR representations. Factorized IR is used for queries which involve many foreign-key-to-foreign-key (FK-FK) joins. Indeed, FK-FK joins often appear specifically in graph workloads due to the typical abundance of many-to-many relationships in graph topologies. For these queries, we employ a two-step query execution process: first, during evaluation, intermediate results are factorized and compressed in a specialized data structure, an *answer graph* (AG [1]); then, the query answer is obtained by defactorizing a resulting AG. Compared to a tuple-block IR which size grows exponentially (in the number of FK-FK joins), AG grows linearly and thus affords a potential dramatic compression of the IR. This, however, comes at a cost of maintenance of the AG (via a sequence of semi-joins or *burn-backs*). Hence, the decision whether to use factorized vs. non-factorized IR is ultimately cost-based.

QuickSilver's execution pipeline includes low-level optimizations such as vectorization and query compilation. For long-running analytical queries, we propose an abstraction model for reasoning about compiled query plans and show how an *entire* query can be compiled using this model [15]. This comes at a compilation cost, however. For shorter-running queries, a subset of query predicates is compiled by using an adaptation of a recently proposed *copy-and-patch* framework [18].

Details and experimental evaluation. Extensive experimental evaluation and further details of our proposed optimizations are available in: BallPark cardinality estimator [17]; Magellan planner subsumes WaveGuide [19, 20] and α, μ-RA [9, 13]; QuickSilver's vectorization and compilation framework [15], multi-threaded (parallel) query execution [16], IR factorization [1], and temporal query processing [21].

3 DEMONSTRATION

We propose a demonstration plan that aims to (1) provide attendees with *deep-insights into the critical challenges* of evaluation of graph-based workloads and (2) *showcase novel solutions* to these challenges as implemented in AvantGraph.

The demonstration relies on our tune-and-explain TunEx framework (shown in Fig. 1). The goal of TunEx is to provide a *multi-faceted* and comprehensive view into the operational insights of the AvantGraph engine and performance characteristics of the corresponding underlying algorithms. TunEx incorporates three complimentary facets of query execution: a fine-grained performance tracer $\boxed{PT}$ which profiles per-thread code execution and presents an execution stack as a detailed flame chart; an execution plan explainer and analyzer $\boxed{PEA}$, a coarser-grained profiler, which presents a cyclic plan, its operators annotated with cardinalities (estimates and real), work-unit distribution, and total time spent processing the operator; and, finally, an execution visualizer $\boxed{EV}$, which visualizes query execution as a graph search, showing real-time exploration of a property graph and highlighting the corresponding bottlenecks during the graph exploration.

We use a diverse collection of real-world and synthetic datasets from encyclopedic (DBPedia[6], YAGO4[7]), investigative (The Bahamas Leaks[8]), life-sciences (UNIPROT[9]), and social network (LDBC datagen [11]) domains. We use *hand-crafted* queries for DBPedia, YAGO4 (used in [1, 9, 20]) and for LDBC (used in the recent LSQB benchmark [11]). We also use queries which were *mined* from a dataset according to a given query shape (e.g., chain, star, snowflake, etc.). We obtain popular and challenging query shapes from a recent query-log study [6].

Scenario set CHS: *Challenges of graph query evaluation.* In this scenario set, we interactively illustrate to attendees the challenges which are *specific* to evaluation of queries on property graphs.

CHS-1: Queries with many FK-FK joins. These are found in all of our datasets, especially for complex shapes (e.g., snowflake). We use mined queries of complex shapes along with hand-crafted queries. QuickSilver is set to use tuple-block IR. During execution, real-time IR blow-up is observed in $\boxed{EV}$, which is later confirmed in $\boxed{PEA}$. Performance degradation is seen in $\boxed{PT}$.

CHS-2: Queries that mix navigation and subgraph matching. We use LSQB queries augmented with navigation and mined queries which combine shapes (e.g., star + chain). Cyclic plans are disabled in Magellan, and the TC operator is used instead. $\boxed{EV}$ shows multiple independent searches exploring large parts of a graph. High-cardinality tuple flow and slow execution is shown in $\boxed{PEA}$.

CHS-3: Queries with diverse and correlated property predicates. We use hand-crafted queries and queries from LSQB. BallPark is set to use simple topological synopses and fall back to independence assumptions. Large errors in cardinality are observed in $\boxed{PEA}$ which lead to bad plans.

Scenario set SOL: *Solutions in AvantGraph.* In this scenario set, we showcase how optimizations proposed and implemented in AvantGraph can solve the challenges identified in ***CHS***.

SOL-1: QuickSilver is set to use a factorized IR. During execution, IR blow-up is controlled as confirmed by $\boxed{PEA}$ and $\boxed{PT}$.

[6]http://dbpedia.org/
[7]http://yago- knowledge.org/resource/
[8]https://www.icij.org/tags/bahamas-leaks/
[9]http://www.uniprot.org/

SOL-2: Magellan is set to use cyclic plans. $\boxed{EV}$ shows multiple interdependent constrained searches exploring the graph. $\boxed{PEA}$ confirms low-cardinality tuple flow.

SOL-3: BallPark is set to use MD histograms and ML-based estimation. $\boxed{PEA}$ shows dramatically improved cardinality estimates leading to better plans.

Attendees will also be able to easily submit their own queries and interact with the live system. Through these rich demonstration scenarios attendees will gain deep insight into the critical challenges of graph query processing and the state of the art innovations realised in AvantGraph to address these challenges.

ACKNOWLEDGMENTS

We sincerely thank Xuming Meng, Oskar van Rest, Kaijie Zhu, Stanley Clark, Mike de Brouwer, Job van der Looij, and Kaj Wortel for their valuable contributions to the AvantGraph project.

REFERENCES

[1] Zahid Abul-Basher, Nikolay Yakovets, Parke Godfrey, Stanley Clark, and Mark Chignell. 2021. Answer graph: Factorization matters in large graphs. In *Proceedings of the 24th International Conference on Extending Database Technology.*
[2] Renzo Angles. 2018. The Property Graph Database Model.. In *AMW.*
[3] R Angles, M Arenas, P Barceló, P Boncz, G Fletcher, C Gutierrez, T Lindaaker, M Paradies, S Plantikow, J Sequeda, et al. 2018. G-CORE: A core for future graph query languages. In *SIGMOD.*
[4] Diego Arroyuelo, Aidan Hogan, Gonzalo Navarro, Juan L Reutter, Javiel Rojas-Ledesma, and Adrián Soto. 2021. Worst-case optimal graph joins in almost no space. In *Proceedings of the 2021 International Conference on Management of Data.*
[5] Angela Bonifati, George Fletcher, Hannes Voigt, and Nikolay Yakovets. 2018. Querying graphs. *Synthesis Lectures on Data Management* 10, 3 (2018), 1–184.
[6] Angela Bonifati, Wim Martens, and Thomas Timm. 2020. An analytical study of large SPARQL query logs. *VLDB J.* 29, 2-3 (2020), 655–679.
[7] Nieves R Brisaboa, Ana Cerdeira-Pena, Guillermo de Bernardo, and Gonzalo Navarro. 2017. Compressed representation of dynamic binary relations with applications. *Information Systems* 69 (2017), 106–123.
[8] Michael Freitag, Maximilian Bandle, Tobias Schmidt, Alfons Kemper, and Thomas Neumann. 2020. Adopting worst-case optimal joins in relational database systems. *Proceedings of the VLDB Endowment* 13, 12 (2020), 1891–1904.
[9] L Jachiet, P Genevès, N Gesbert, and N Layaïda. 2020. On the optimization of recursive relational queries: Application to graph queries. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data.* 681–697.
[10] A.O. Mendelzon and P.T. Wood. 1995. Finding regular simple paths in graph databases. *SIAM J. Comput.* 24, 6 (1995), 1235–1258.
[11] Amine Mhedhbi, Matteo Lissandrini, Laurens Kuiper, Jack Waudby, and Gábor Szárnyas. 2021. LSQB: a large-scale subgraph query benchmark. In *Proceedings of the 4th ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA).* 1–11.
[12] Thomas Neumann and Michael J Freitag. 2020. Umbra: A Disk-Based System with In-Memory Performance.. In *CIDR.*
[13] C. Ordonez. 2010. Optimization of linear recursive queries in SQL. *Knowledge and Data Engineering, IEEE Transactions on* 22, 2 (2010), 264–277.
[14] J Patel, H Deshmukh, J Zhu, N Potti, Z Zhang, M Spehlmann, H Memisoglu, and S Saurabh. 2018. Quickstep: A data platform based on the scaling-up approach. *Proceedings of the VLDB Endowment* 11, 6 (2018), 663–676.
[15] B. van de Wall. 2020. Fully Compiled Execution of Conjunctive Graph Queries. MSc Thesis, TU Eindhoven.
[16] J. van der Looij. 2021. Avantstep: parallel query execution in graph databases. MSc Thesis, TU Eindhoven.
[17] Wilco van Leeuwen, George Fletcher, and Nikolay Yakovets. 2022. A General Cardinality Estimation Framework for Subgraph Matching in Property Graphs. *Knowledge and Data Engineering, IEEE Transactions on* (2022).
[18] Haoran Xu and Fredrik Kjolstad. 2021. Copy-and-patch compilation: a fast compilation algorithm for high-level languages and bytecode. *Proceedings of the ACM on Programming Languages* 5, OOPSLA (2021), 1–30.
[19] Nikolay Yakovets. 2017. Optimization of Regular Path Queries in Graph Databases. PhD Dissertation, York University.
[20] Nikolay Yakovets, Parke Godfrey, and Jarek Gryz. 2016. Query Planning for Evaluating SPARQL Property Paths. In *SIGMOD.* San Francisco, 1875–1889.
[21] Kaijie Zhu, George Fletcher, and Nikolay Yakovets. 2021. Leveraging temporal and topological selectivities in temporal-clique subgraph query processing. In *2021 IEEE 37th International Conference on Data Engineering (ICDE).* IEEE.