Modern Techniques For Querying Graph-structured Databases

Modern Techniques For Querying Graph-structured Databases

Amine Mhedhbi
Polytechnique Montreal
amine.mhedhbi@polymtl.ca

Amol Deshpande
University of Maryland
amol@umd.edu

Semih Salihoğlu
University of Waterloo
semih.salihoglu@uwaterloo.ca

now
the essence of knowledge
Boston — Delft

Foundations and Trends® in Databases

Published, sold and distributed by:
now Publishers Inc.
PO Box 1024
Hanover, MA 02339
United States
Tel. +1-781-985-4510
www.nowpublishers.com
sales@nowpublishers.com

Outside North America:
now Publishers Inc.
PO Box 179
2600 AD Delft
The Netherlands
Tel. +31-6-51115274

The preferred citation for this publication is

A. Mhedhbi *et al.*. *Modern Techniques For Querying Graph-structured Databases*. Foundations and Trends® in Databases, vol. 14, no. 2, pp. 72–185, 2024.

ISBN: 978-1-63828-425-3

Foundations and Trends® in Databases
Volume 14, Issue 2, 2024

Editorial Scope

Foundations and Trends® in Databases publishes survey and tutorial articles in the following topics:

- Data Models and Query Languages
- Query Processing and Optimization
- Storage, Access Methods, and Indexing
- Transaction Management, Concurrency Control and Recovery
- Deductive Databases
- Parallel and Distributed Database Systems
- Database Design and Tuning
- Metadata Management
- Object Management
- Trigger Processing and Active Databases
- Data Mining and OLAP
- Approximate and Interactive Query Processing
- Data Warehousing
- Adaptive Query Processing
- Data Stream Management
- Search and Query Integration
- XML and Semi-Structured Data
- Web Services and Middleware
- Data Integration and Exchange
- Private and Secure Data Management
- Peer-to-Peer, Sensornet and Mobile Data Management
- Scientific and Spatial Data Management
- Data Brokering and Publish/-Subscribe
- Data Cleaning and Information Extraction
- Probabilistic Data Management

Information for Librarians

Foundations and Trends® in Databases, 2024, Volume 14, 4 issues. ISSN paper version 1931-7883. ISSN online version 1931-7891. Also available as a combined paper and online subscription.

Contents

Modern Techniques For Querying Graph-structured Databases

Amine Mhedhbi[1], Amol Deshpande[2] and Semih Salihoğlu[3]

[1] *Polytechnique Montreal, Canada; amine.mhedhbi@polymtl.ca*
[2] *University of Maryland, USA; amol@cs.umd.edu*
[3] *University of Waterloo, Canada; semih.salihoglu@uwaterloo.ca*

ABSTRACT

In an era of increasingly interconnected information, graph-structured data has become pervasive across numerous domains from social media platforms and telecommunication networks to biological systems and knowledge graphs. However, traditional database management systems often struggle when confronted with the unique challenges posed by graph-structured data, in large part due to the explosion of intermediate results, the complexity of join-heavy queries, and the use of regular path queries.

This survey provides a comprehensive overview of *modern* query processing techniques designed to address these challenges. We focus on four key components that have emerged as pivotal in optimizing queries on graph-structured databases: (1) Predefined joins, which leverage precomputed data structures to accelerate joins; (2) Worst-case optimal join algorithms, that avoid redundant computations for queries with cycles; (3) Factorized representations, which compress intermediate and final query results; and (4) Advanced techniques for processing recursive queries, essential for traversing graph structures. For each component,

Amine Mhedhbi, Amol Deshpande and Semih Salihoğlu (2024), “Modern Techniques For Querying Graph-structured Databases”, Foundations and Trends® in Databases: Vol. 14, No. 2, pp 72–185. DOI: 10.1561/1900000090.

we delve into its theoretical underpinnings, explore design considerations, and discuss the implementation challenges associated with integrating these techniques into existing database management systems. This survey aims to serve as a comprehensive resource for both researchers pushing the boundaries of query processing and practitioners seeking to implement state-of-the-art techniques, in addition to offering insights into future research directions in this rapidly evolving field.

1 Introduction

Graph-structured data is ubiquitous in many real-world domains and applications. Social networks, financial transactions, telecommunication networks, and knowledge graphs are just a few examples where data is naturally represented as a graph with entities as *nodes* and relationships as *edges*. Querying and analyzing these graph-structured datasets is integral to a wide range of analytical applications such as recommendations in social networks, fraud detection in financial transaction networks, threat detection in call networks, and inference over knowledge bases (Sahu *et al.*, 2020).

As an example application domain, consider financial transaction networks. These networks model entities like individuals, businesses, and financial institutions as nodes, with transactions between them as edges. Analyzing these networks is crucial for tasks like detecting money laundering, tax evasion, and other financial crimes. Typical queries involve finding long chains of transactions (acyclic paths) that may indicate suspicious activity, or identifying tightly connected clusters (cliques/near-cliques) of entities engaging in coordinated illicit behavior.

Figure 1.1 shows an example graph containing 6 entities (nodes) and 7 relationships (edges) between them. We assume that the input

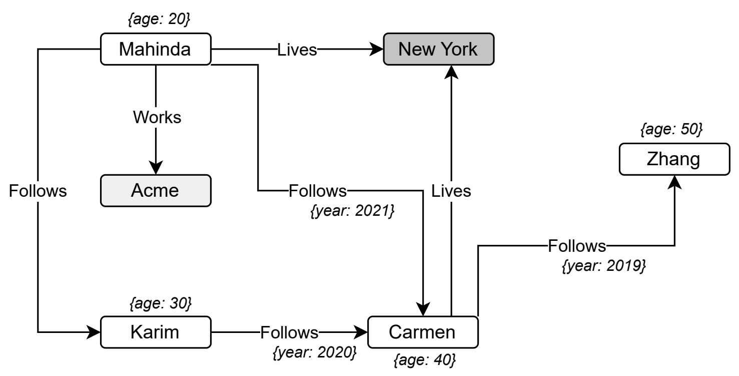

Figure 1.1: An example of a *property graph* capturing social network data. *Name* properties of nodes are written directly inside the rectangles representing nodes.

Person

name	age
Mahinda	20
Karim	30
Carmen	40
Zhang	50

Follows

pFrom	pTo	year
Mahinda	Karim	2021
Carmen	Zhang	2019
Mahinda	Carmen	2021
Karim	Carmen	2020
Mahinda	Zhang	2021

Lives

pName	lName
Mahinda	New York
Carmen	New York

Works

pName	cName
Mahinda	Acme

Locations

name
New York

Companies

name
Acme

Figure 1.2: Representation of the example graph from Figure 1.1 as a collection of relations.

graphs are directed in this monograph, however, most of the discussion applies to undirected graphs as well. Further, we logically model these graphs as a collection of relations as shown in Figure 1.2, where there is a separate relation for every entity type (label) and every relationship type (label). We note that this is a logical representation that we adopt for presentation purposes, and does not necessarily dictate the physical storage layouts, the data structures and indexes that may be built on top, or the query processing algorithms. In other words, we use the relational representation to express the queries to be executed against this graph so that we can more easily contrast and compare with relational query processing techniques; however, following the principle of *physical data independence*, the physical storage representation is not required to match that logical representation.

Given the tabular representation of graphs, a large fraction of graph analysis and querying tasks (collectively referred to as *graph workloads*) can be seen as select-project-join-aggregate queries on these tables, potentially with recursion. However, unlike typically relational workloads, graph-structured datasets and workloads have two primary defining features:

(i) **Prevalence of many-to-many relations across entities:** Unlike typical relational workloads which primarily feature key-foreign key connections across tables, graph workloads primarily contain many-to-many relationships (e.g., all the relationships in the example graph above, i.e., *Follows*, *Works*, and *Lives*, are many-to-many relationships).

(ii) **Prevalence of complex join-heavy queries over these relations:** The prevalent tasks in graph workloads translate to queries with many joins across these many-to-many relationships. The joins in these queries can have several different structures:

    (i) cyclic, such as when finding cliques of phone calls;
    (ii) acyclic, such as when finding long chains of financial transactions; or
    (iii) recursive, such as when finding shortest connections between users in social networks.

This contrasts with traditional relational workloads, such as those found in the popular TPC benchmarks, that contain many primary-foreign key (`PK-FK`) joins. The combination of complex join structures in the graph workloads and the many-to-many cardinality of relations in these datasets pose serious challenges for traditional query processors. For example, queries can generate large intermediate relations that often cannot be handled by traditional techniques.

The last decade has seen the emergence of numerous prototype and commercial DBMSs that are optimized for graph workloads. These include specialized systems such as graph DBMSs (GDBMSs) that adopt the property graph data model, e.g., Neo4j (Neo4j, Inc, 2023a), TigerGraph (Tigergraph, 2023), GraphflowDB (Kankanamge *et al.*, 2017), Kùzu (Feng *et al.*, 2023), Avantgraph (Leeuwen *et al.*, 2022); earlier RDF systems, e.g., RDF-3x (Neumann and Weikum, 2010); and graph-optimized extensions of RDBMSs, e.g., GR-Fusion (Hassan *et al.*, 2018), GRainDB (Jin and Salihoglu, 2022), and GQ-Fast (Lin *et al.*, 2016). The goal of this monograph is to survey a set of *modern* query processing techniques that have been integrated into the query processors of these systems. These include:

(i) *Pointer-based joins* (Section 2) that rely on system-level dense IDs in contrast to traditional value-based joins in RDBMSs.

(ii) *Worst-case optimal join (WCOJ) algorithms* (Section 3), which are a new class of join algorithms that address the problem of large intermediate size generation for cyclic many-to-many join queries. Compared to traditional plans that use a tree of binary join algorithms and perform the joins in a query pairs of table at a time, WCOJs algorithms perform the joins one column at a time.

(iii) *Factorization* (Section 4), a class of techniques to compress intermediate relations that exhibit *multi-valued dependencies* generated when performing many-to-many joins, specifically in the acyclic parts of queries. The theory of factorization represents such intermediate relations in different *factorized representations* as unions of Cartesian products instead of flat tuples. The theory of factorization explains when query processors can exploit such factorized representations by analyzing the join conditions between variables during query compilation time.

(iv) *Techniques for regular path queries* (Section 5), a popular class of recursive graph queries. Amongst recursive queries, most prior work focuses on regular path queries. We cover the traditional automata-based plans of Mendelzon and Wood (1989) and α-join plans based on α relational algebra introduced by Agrawal (1988), and the more recent WaveGuide plans of Yakovets *et al.* (2016a) that mix both of these approaches.

We overview: (i) the foundations of these techniques when appropriate; (ii) the current design choices different DBMSs have made to integrate these techniques; and (iii) the challenges for existing implementation approaches, which provide promising avenues for further research. Our goal is to bring a structure to this vast theory and systems-oriented literature. We focus on the application of these techniques for join processing over static databases though we briefly mention works that apply these techniques to the problem of incrementally maintaining query results when the underlying databases are dynamic.

We primarily cover these techniques in the context of a centralized, sequential computation model, which has been the focus of most of the work on these techniques so far, including WCOJ algorithms, factorization, and RPQs. There is a vast body of work on building on parallel and distributed graph analytics platforms (Yan *et al.*, 2017), primarily based on the so-called *vertex-centric programming framework* (Malewicz *et al.*, 2010). This framework has also been incorporated into several relational database systems (Fan *et al.*, 2015; Jindal *et al.*, 2014). However, the target workload for those systems is very different from the types of queries that we focus on in this monograph. Some of the example graph analysis tasks include: finding most central or influential nodes in the graph (e.g., by calculating metrics such *page rank* or *betweenness centrality*), identifying communities in the graph, understanding influence propagation, etc. Although there is some overlap, the algorithmic techniques discussed in this monograph are not applicable to executing those types of tasks; instead, specialized parallel systems (Shun and Blelloch, 2013; Wang *et al.*, 2016) are typically used given the scale of the graphs involved in the application domains like social media, finance, disease transmission, etc. On the flip side, it has been shown how to implement multi-way joins efficiently on top of the vertex-centric framework (Smagulova and Deutsch, 2021) and other distributed programming paradigms (Afrati and Ullman, 2011); we discuss some of that work briefly where appropriate.

The techniques we discuss in this monograph are generally applicable to any database system where the workload maps to multi-way join queries over relations. This includes RDF databases that store the RDF data as a collection of tables and map queries over the data (typically in SPARQL) to multi-way join queries over those tables (Neumann and Weikum, 2010; Abadi *et al.*, 2007; Erling and Mikhailov, 2009). It also includes document databases that adopt XML or JSON data model, but "shred" the data into a collection of tables and translate the queries to join queries (Tatarinov *et al.*, 2002). However, they do not apply directly to systems that use specialized storage schemes or index data structures to execute queries (e.g., gStore, Zou *et al.*, 2014a). We note that there are a number of similarities between WCOJ algorithms and traversal-based techniques for subgraph pattern matching (Sun *et al.*, 2020), but more work is needed to unify these somewhat disparate lines of work.

Finally, we focus exclusively on read queries in this monograph. In most cases, updates to the graph can be directly mapped to updates to the underlying tables, and can benefit from the mature and efficient support for transactions and ACID properties in relational databases. This is, in fact, a key motivation behind using a relational database as the backend storage for a graph database. However, the specific mix of queries and updates may influence the decisions about how many and which tables to use. We don't discuss these issues further in this monograph.

1.1 Target Audience

This monograph is intended for readers who are familiar with basic concepts of internals of databases, such as the general paradigm of compiling high-level queries into executable query plans and core query processing operators, such as scans and joins. Beyond that we cover the necessary background in each section. We are particularly interested in making the monograph accessible to readers with graph analytics or graph processing systems backgrounds. However, we adopt a relational view of query evaluation even if logically the datasets in our examples are often modeled as graphs. This is because the foundations of many of the techniques we cover were developed in the context of join processing over sets of records. At parts, we cover some advanced material on database theory. We accompany these parts with suggestions to skip for readers who may be more interested in understanding the core query processing techniques and how they are integrated into systems.

1.2 Brief Background

We end this introductory section with a brief overview of some background on the formal notation we use for describing join queries and databases. The necessary notation for regular path queries is covered in Section 5. Background for each of the techniques we cover is provided in detail in each section.

Unless otherwise stated, we assume that input graphs are directed and modeled as binary relations, that are denoted with capital letters R and E or variants such as R_1 or E_2. The attributes of relations are generally denoted with lowercase letters that start with a, such as a_1 or a_2. In some figures, we use attributes 'from' and 'to', 'src' or 'dst', or a similar variant of these words to denote the sources and destinations of the edges.

We consider natural join queries over these binary relations that we denote by Q (or a variant Q_i). We generally assume full join queries, i.e., where no projections occur. We denote queries in Datalog syntax, where we generally omit the attributes in the head of the rules. The following is an example showing how we denote the "triangle" query:

$$Q_{\Delta} := R_1(a_1, a_2), R_2(a_2, a_3) R_3(a_3, a_1)$$

Sometimes, we write a predicate next to a variable in the head or body of these rules to indicate filters. For example, $Q(a_1{=}1, a_2, a_3) :=$ $R_1(a_1 = 1, a_2)$,$R_2(a_2, a_3)$, $R_3(a_3, a_1 = 1)$ represents the query that finds all triangles where a_1 has value 1.

In a few parts of the monograph, we also use example queries from SQL and the Cypher query language (Francis *et al.*, 2018). Cypher is the query language of the Neo4j system that is also adopted by several other GDBMSs, such as MemGraph (Memgraph Ltd, 2023) and Kùzu (Feng *et al.*, 2023). The meanings of Cypher queries is explained in the parts of text when they are used.

References

Abadi, D. J., A. Marcus, S. R. Madden, and K. Hollenbach. (2007). "Scalable semantic web data management using vertical partitioning". In: *Proceedings of the 33rd international conference on Very large data bases*. 411–422.

Aberger, C. R., A. Lamb, S. Tu, A. Nötzli, K. Olukotun, and C. Ré. (2017). "EmptyHeaded: A Relational Engine for Graph Processing". *TODS*. 42(4).

Abo Khamis, M., H. Q. Ngo, and D. Suciu. (2016). "Computing Join Queries with Functional Dependencies". In: *ACM PODS*.

Abul-Basher, Z., N. Yakovets, P. Godfrey, S. Clark, and M. H. Chignell. (2021). "Answer Graph: Factorization Matters in Large Graphs". In: *EDBT*. Ed. by Y. Velegrakis, D. Zeinalipour-Yazti, P. K. Chrysanthis, and F. Guerra. OpenProceedings.org.

Afrati, F. N. and J. D. Ullman. (2011). "Optimizing Multiway Joins in a Map-Reduce Environment". *IEEE Transactions on Knowledge and Data Engineering*. 23(9).

Agrawal, R. (1988). "Alpha: an extension of relational algebra to express a class of recursive queries". *IEEE Transactions on Software Engineering*. 14(7).

Ahmad, Y., O. Kennedy, C. Koch, and M. Nikolic. (2012). "DBToaster: Higher-order Delta Processing for Dynamic, Frequently Fresh Views". *PVLDB*. 5(10).

Ammar, K., F. McSherry, S. Salihoglu, and M. Joglekar. (2018). "Distributed Evaluation of Subgraph Queries Using Worst-case Optimal Low-memory Dataflows". *PVLDB*. 11(6).

Aref, M., B. ten Cate, T. J. Green, B. Kimelfeld, D. Olteanu, E. Pasalic, T. L. Veldhuizen, and G. Washburn. (2015). "Design and Implementation of the LogicBlox System". In: *ACM SIGMOD*.

Arroyuelo, D., A. Hogan, G. Navarro, and J. Rojas-Ledesma. (2022). "Time-and space-efficient regular path queries". In: *ICDE*.

Atserias, A., M. Grohe, and D. Marx. (2008). "Size Bounds and Query Plans for Relational Joins". In: *FOCS*.

Bachman, C. W. (2009). "The Origin of the Integrated Data Store (IDS): The First Direct-Access DBMS". *IEEE Annals of the History of Computing*. 31(4).

Bakibayev, N., T. Kociský, D. Olteanu, and J. Zavodny. (2013). "Aggregation and Ordering in Factorised Databases". *PVLDB*. 6(14).

Bakibayev, N., D. Olteanu, and J. Zavodny. (2012). "FDB: A Query Engine for Factorised Relational Databases". *PVLDB*. 5(11).

Beame, P., P. Koutris, and D. Suciu. (2017). "Communication Steps for Parallel Query Processing". *Journal of the ACM*. 64(6). DOI: 10.1145/3125644.

Bhattarai, B., H. Liu, and H. H. Huang. (2019). "CECI: Compact Embedding Cluster Index for Scalable Subgraph Matching". In: *SIGMOD*.

Bi, F., L. Chang, X. Lin, L. Qin, and W. Zhang. (2016). "Efficient Subgraph Matching by Postponing Cartesian Products". In: *SIGMOD*.

Blakeley, J. A., P.-A. Larson, and F. W. Tompa. (1986). "Efficiently Updating Materialized Views". *SIGMOD Record*. 15(2).

Boncz, P. A., M. Zukowski, and N. Nes. (2005). "MonetDB/X100: Hyper-Pipelining Query Execution". In: *CIDR*.

Bonifati, A., G. Fletcher, H. Voigt, N. Yakovets, and H. V. Jagadish. (2018). *Querying Graphs*. Morgan & Claypool Publishers.

Cai, W., M. Balazinska, and D. Suciu. (2019). "Pessimistic Cardinality Estimation: Tighter Upper Bounds for Intermediate Join Cardinalities". In: *SIGMOD*.

Chen, J., Y. Huang, M. Wang, S. Salihoglu, and K. Salem. (2022). "Accurate Summary-Based Cardinality Estimation through the Lens of Cardinality Estimation Graphs". *PVLDB*. 15(8).

Codd, E. F. (1982). "Relational Database: A Practical Foundation for Productivity". *CACM*. 25(2).

Delobel, C. (1978). "Normalization and hierarchical dependencies in the relational data model". *TODS*. 3(3).

Dey, S., V. Cuevas-Vicenttín, S. Köhler, E. Gribkoff, M. Wang, and B. Ludäscher. (2013). "On implementing provenance-aware regular path queries with relational query engines". In: *EDBT/ICDT Workshops*.

Erling, O. and I. Mikhailov. (2009). "Virtuoso: RDF support in a native RDBMS". In: *Semantic web information management: a model-based perspective*. Springer. 501–519.

Fagin, R. (1977). "Multivalued dependencies and a new normal form for relational databases". *TODS*. 2(3).

Fan, J., A. G. S. Raj, and J. M. Patel. (2015). "The Case Against Specialized Graph Analytics Engines." In: *CIDR*.

Farias, B., C. Rojas, and D. Vrgoc. (2023). "Evaluating Regular Path Queries in GQL and SQL/PGQ: How Far Can The Classical Algorithms Take Us?" *CoRR*. abs/2306.02194.

Feng, X., G. Jin, Z. Chen, C. Liu, and S. Salihoğlu. (2022). "Kùzu Database Management System Source Code". URL: https://github.com/kuzudb/kuzu.

Feng, X., G. Jin, Z. Chen, C. Liu, and S. Salihoğlu. (2023). "Kùzu Graph Database Management System". In: *CIDR*.

Francis, N., A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor. (2018). "Cypher: An Evolving Query Language for Property Graphs". In: *ACM SIGMOD*.

Freitag, M., M. Bandle, T. Schmidt, A. Kemper, and T. Neumann. (2020). "Adopting Worst-Case Optimal Joins in Relational Database Systems". *PVLDB*. 13(12).

Graefe, G. (1994). "Volcano - An Extensible and Parallel Query Evaluation System". *TKDE*. 6(1).

Gupta, P., A. Mhedhbi, and S. Salihoglu. (2021). "Columnar Storage and List-based Processing for Graph Database Management Systems". *PVLDB*. 14(11).

Han, M., H. Kim, G. Gu, K. Park, and W. Han. (2019). "Efficient Subgraph Matching: Harmonizing Dynamic Programming, Adaptive Matching Order, and Failing Set Together". In: *SIGMOD*.

Hassan, M. S., T. Kuznetsova, H. C. Jeong, W. G. Aref, and M. Sadoghi. (2018). "Extending In-Memory Relational Database Engines with Native Graph Support". In: *EDBT*.

Huang, Z. and E. Wu. (2023). "Lightweight Materialization for Fast Dashboards Over Joins". *SIGMOD*. 1(4).

Idreos, S., M. L. Kersten, and S. Manegold. (2007). "Database Cracking". In: *Conference on Innovative Data Systems Research*.

ISO/IEC JTC 1/SC 32. (2024). "SQL/PGQ Standard". URL: https://www.iso.org/standard/79473.html.

JCC Consulting, Inc. (2024). "GQL Standard". URL: https://www.gqlstandards.org/.

Jin, G. and S. Salihoglu. (2022). "Making RDBMSs Efficient on Graph Workloads Through Predefined Joins". *PVLDB*. 15(5).

Jindal, A., P. Rawlani, E. Wu, S. Madden, A. Deshpande, and M. Stonebraker. (2014). "Vertexica: your relational friend for graph analytics!" *Proceedings of the VLDB Endowment*. 7(13): 1669–1672.

Joglekar, M. and C. Ré. (2018). "It's All a Matter of Degree - Using Degree Information to Optimize Multiway Joins". *Theory of Computing Systems*. 62(4).

Kalinsky, O., Y. Etsion, and B. Kimelfeld. (2017). "Flexible Caching in Trie Joins". In: *EDBT*. Ed. by V. Markl, S. Orlando, B. Mitschang, P. Andritsos, K. Sattler, and S. Breß.

Kankanamge, C., S. Sahu, A. Mhedhbi, J. Chen, and S. Salihoglu. (2017). "Graphflow: An Active Graph Database". In: *SIGMOD*.

Kara, A., M. Nikolic, D. Olteanu, and H. Zhang. (2023). "F-IVM: Analytics over Relational Databases under Updates". *CoRR*. abs/2303.08583.

Khamis, M. A., H. Q. Ngo, C. Ré, and A. Rudra. (2016). "Joins via Geometric Resolutions: Worst Case and Beyond". *TODS*. 41(4).

Koschmieder, A. and U. Leser. (2012). "Regular path queries on large graphs". In: *SSDBM*. Springer.

Koutris, P., S. Salihoglu, and D. Suciu. (2018). "Algorithmic Aspects of Parallel Data Processing". *Foundations and Trends® in Databases*. 8(4).

Lapaugh, A. and C. Papadimitriou. (1984). "The even-path problem for graphs and digraphs". *Networks*. 14.

Leeuwen, W. v., T. Mulder, B. van de Wall, G. Fletcher, and N. Yakovets. (2022). "AvantGraph Query Processing Engine". *PVLDB*. 15(12).

Leis, V., B. Radke, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann. (2018). "Query Optimization through the Looking Glass, and What We Found Running the Join Order Benchmark". *VLDBJ*. 27(5).

Lin, C., B. Mandel, Y. Papakonstantinou, and M. Springer. (2016). "Fast In-Memory SQL Analytics on Typed Graphs". In: *ICDE*.

Losemann, K. and W. Martens. (2013). "The complexity of regular expressions and property paths in SPARQL". *TODS*. 38(4).

Malewicz, G., M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. (2010). "Pregel: a system for large-scale graph processing". In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. 135–146.

Memgraph Ltd. (2023). "MemGraph". URL: https://memgraph.com/.

Mendelzon, A. O. and P. T. Wood. (1995). "Finding regular simple paths in graph databases". *SIAM J. Comput.* 24(6).

Mendelzon, A. O. and P. T. Wood. (1989). "Finding Regular Simple Paths in Graph Databases". *SIAM J. Comput.* 24: 1235–1258.

Mhedhbi, A. (2023). "GraphflowDB: Scalable Query Processing on Graph-Structured Relations". *PhD thesis*. URL: http://hdl.handle.net/10012/19981.

Mhedhbi, A., C. Kankanamge, and S. Salihoglu. (2021). "Optimizing One-time and Continuous Subgraph Queries using Worst-case Optimal Joins". *TODS*. 46(2).

Mhedhbi, A. and S. Salihoglu. (2019). "Optimizing Subgraph Queries by Combining Binary and Worst-Case Optimal Joins". *PVLDB*. 12(11).

Neo4j, Inc. (2023a). "Neo4j". URL: https://neo4j.com/.

Neo4j, Inc. (2023b). "Neo4j Record Design". URL: https://neo4j.com/developer/kb/understanding-data-on-disk/.

Neumann, T. and M. J. Freitag. (2020). "Umbra: A Disk-Based System with In-Memory Performance". In: *CIDR*.

Neumann, T. and G. Weikum. (2010). "The RDF-3X engine for scalable management of RDF data". In: *VLDBJ*.

Ngo, H. Q., D. T. Nguyen, C. Re, and A. Rudra. (2014). "Beyond Worst-Case Analysis for Joins with Minesweeper". In: *PODS*.

Ngo, H. Q., E. Porat, C. Ré, and A. Rudra. (2012). "Worst-case Optimal Join Algorithms: [Extended Abstract]". In: *PODS*.

Ngo, H. Q., C. Ré, and A. Rudra. (2013). "Skew strikes back: new developments in the theory of join algorithms". In: *SIGMOD Rec.*

Nguyen, V.-Q. and K. Kim. (2017). "Efficient regular path query evaluation by splitting with unit-subquery cost matrix". *IEICE Transactions on Information and Systems*. 100(10).

Nikolic, M., H. Zhang, A. Kara, and D. Olteanu. (2020). "F-IVM: Learning over Fast-Evolving Relational Data". In: *ACM SIGMOD*.

Olteanu, D. and M. Schleich. (2016). "Factorized Databases". *SIGMOD Rec.* 45(2).

Olteanu, D. and J. Zavodny. (2015). "Size Bounds for Factorised Representations of Query Results". *TODS*. 40(1).

Peng, Y., Y. Zhang, X. Lin, L. Qin, and W. Zhang. (2020). "Answering billion-scale label-constrained reachability queries within microsecond". *PVLDB*. 13(6).

Raasveldt, M. and H. Mühleisen. (2019a). "DuckDB: An Embeddable Analytical Database". In: *SIGMOD*.

Raasveldt, M. and H. Mühleisen. (2019b). "DuckDB: an Embeddable Analytical Database". In: *SIGMOD*.

Sahu, S., A. Mhedhbi, S. Salihoglu, J. Lin, and M. T. Özsu. (2020). "The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing: Extended Survey". 13(12).

Schleich, M. and D. Olteanu. (2020). "LMFAO: An Engine for Batches of Group-by Aggregates: Layered Multiple Functional Aggregate Optimization". 13(12).

Shun, J. and G. E. Blelloch. (2013). "Ligra: a lightweight graph processing framework for shared memory". In: *Proceedings of the 18th ACM SIGPLAN symposium on Principles and practice of parallel programming*. 135–146.

Silberschatz, A., H. Korth, and S. Sudarshan. (2005). *Database Systems Concepts*. 5th ed. McGraw-Hill, Inc.

Smagulova, A. and A. Deutsch. (2021). "Vertex-centric Parallel Computation of SQL Queries". In: *Proceedings of the 2021 International Conference on Management of Data*. 1664–1677.

Sun, S., X. Sun, Y. Che, Q. Luo, and B. He. (2020). "Rapidmatch: A holistic approach to subgraph query processing". *Proceedings of the VLDB Endowment*. 14(2): 176–188.

Tatarinov, I., S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang. (2002). "Storing and querying ordered XML using a relational database system". In: *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. 204–215.

Then, M., M. Kaufmann, F. Chirigati, T.-A. Hoang-Vu, K. Pham, A. Kemper, T. Neumann, and H. T. Vo. (2014). "The More the Merrier: Efficient Multi-source Graph Traversal". *PVLDB*. 8(4).

Tigergraph. (2023). "TigerGraph". URL: https://www.tigergraph.com/.

Valduriez, P. (1987). "Join Indices". *ACM TODS*. 12(2).

Valstar, L. D., G. H. Fletcher, and Y. Yoshida. (2017). "Landmark Indexing for Evaluation of Label-Constrained Reachability Queries". In: *SIGMOD*.

Veldhuizen, T. L. (2012). "Leapfrog Triejoin: a worst-case optimal join algorithm". *CoRR*. abs/1210.0481.

Veldhuizen, T. L. (2013). "Incremental Maintenance for Leapfrog Triejoin". *CoRR*. abs/1303.5313.

W3C. (2024). "SPARQL Standard". URL: https://www.w3.org/TR/sparql11-query/.

Wang, Y., A. Davidson, Y. Pan, Y. Wu, A. Riffel, and J. D. Owens. (2016). "Gunrock: A high-performance graph processing library on the GPU". In: *Proceedings of the 21st ACM SIGPLAN symposium on principles and practice of parallel programming*. 1–12.

Wang, Y. R., M. Willsey, and D. Suciu. (2023). "Free Join: Unifying Worst-Case Optimal and Traditional Joins". In: *SIGMOD*.

Wolde, D. ten, T. Singh, G. Szarnyas, and P. Boncz. (2023). "DuckPGQ: Efficient Property Graph Queries in an analytical RDBMS". In: *CIDR*.

Xirogiannopoulos, K. and A. Deshpande. (2017). "Extracting and analyzing hidden graphs from relational databases". In: *SIGMOD*.

Xirogiannopoulos, K., V. Srinivas, and A. Deshpande. (2017). "Graphgen: Adaptive graph processing using relational databases". In: *GRADES-NDA*.

Yakovets, N., P. Godfrey, and J. Gryz. (2013). "Evaluation of SPARQL Property Paths via Recursive SQL". In: *Alberto Mendelzon Workshop on Foundations of Data Management*.

Yakovets, N., P. Godfrey, and J. Gryz. (2016a). "Query Planning for Evaluating SPARQL Property Paths". In: *SIGMOD*.

Yakovets, N., P. Godfrey, and J. Gryz. (2016b). "Query planning for evaluating SPARQL property paths". In: *SIGMOD*.

Yan, D., Y. Bu, Y. Tian, A. Deshpande, *et al.* (2017). "Big graph analytics platforms". *Foundations and Trends® in Databases*. 7(1-2): 1–195.

Yannakakis, M. (1981). "Algorithms for Acyclic Database Schemes". In: *PVLDB*.

Yu, J. X. and J. Cheng. (2010). "Graph Reachability Queries: A Survey". In: *Managing and Mining Graph Data*. Springer US.

Zhu, J., N. Potti, S. Saurabh, and J. M. Patel. (2017). "Looking ahead makes query plans robust: Making the initial case with in-memory star schema data warehouse workloads". *PVLDB*. 10(8).

Zou, L., M. T. Özsu, L. Chen, X. Shen, R. Huang, and D. Zhao. (2014a). "gStore: a graph-based SPARQL query engine". *The VLDB journal*. 23: 565–590.

Zou, L., K. Xu, J. X. Yu, L. Chen, Y. Xiao, and D. Zhao. (2014b). "Efficient Processing of Label-constraint Reachability Queries in Large Graphs". *Information Systems*. 40.

Zukowski, M., M. van de Wiel, and P. A. Boncz. (2012). "Vectorwise: A Vectorized Analytical DBMS". In: *ICDE*.