# Beyond Worst-Case-Optimal Database Joins

FABRIZIO BARISIONE, University of Chile, Chile

The natural join is arguably the most studied operation in database theory and systems. In 2008, the way traditional join algorithms work was found to be suboptimal and a new tight bound to reach worst-case optimality (wco) was proposed. Several wco join algorithms have been designed over the years, but they all experience the same problem: for a relation $R$ with $d$ dimensions, $d!$ indexes are required, which in terms of space makes them impractical for higher dimensions. To solve this limitation, a new data structure dubbed the *ring* was designed to enable worst-case optimality in graph databases using compact space. The experiments showed that the ring outperforms its competitors in space and time.

In this research, we propose to further enhance the ring performance, aiming to reach beyond wco runtime guarantees, and translate them into practical time improvements. We will start by studying, designing and implementing query optimizations for graph databases, where the ring was implemented and tested. Then, we will expand it to support higher dimensions, extending its functionality to relational databases.

Additional Key Words and Phrases: Worst-case Optimal Join Algorithms, AGM Bound, Relational Databases, Graph Databases, Compact Data Structures

## 1 PROBLEM STATEMENT AND STATE OF THE ART

The success of the relational model relies on the fact that efficient ways to resolve relational algebra expressions have been developed and implemented since the 60s. The join (most typically, the natural join) is by far the most expensive of those operations.

Traditional join algorithms such as *merge join*, *hash join* and *nested loop* operate in a binary fashion. That is, they resolve joins by grouping pairs of relations and producing intermediate results in the form of tables. On the other hand, Commercial databases use highly optimized join heuristics to determine the join evaluation order, considering a variety of factors such as the table sizes, memory, statistics, etc. These optimizations enable the binary join algorithms to have better runtimes.

New developments in join algorithms have been made during the last decades, starting with a breakthrough: traditional query plans using a binary join approach were proven to be inherently *suboptimal* [6]. It was shown that there are types of queries for which any binary join plan is slower than the best possible runtime by a polynomial factor in the data size.

In 2008, Atserias, Grohe and Marx (AGM) defined a limit on the number of results for a join query [6], bounding it in terms of the sizes of the participating relations and the structure of the join. Historically, RDBMSs mostly considered the table cardinalities to optimize the join process. Instead, given a query $Q$, Atserias et al. demonstrated that, with the structural information of $Q$, we can upper bound its size more tightly. This discovery led to the concept of multi-way joins, which operate differently from binary joins, by intersecting all common attributes at the same time rather than by pairs.

Several worst-case optimal (wco) join algorithms that reach the so-called AGM bound were devised [12], with Leapfrog Triejoin[14] (LTJ) the most renowned of them due to its simplicity. LTJ guarantees wco runtimes by joining all input relations simultaneously, without yielding intermediate results. The latter is accomplished by storing the rows of each relation in a trie data structure, in which each row sequence can be read along a root-to-leaf path in the trie. The LTJ algorithm performs a backtracking over all those tries simultaneously, binding each variable of the query at a time. This requires, however, that the tries are built over a column ordering of the tables

Author's address: Fabrizio Barisione, University of Chile, Santiago, Chile, fbarisio@dcc.uchile.cl.

that is suitable for that particular query, which in practice implies that a different trie must be built for each of the $d!$ orders of the $d$ columns of each table.

Worst-case optimal join algorithms are particularly promising for graph databases [14] because these are modeled as relations with low fixed arity (e.g. RDF [1]), and thus graph queries tend to feature many joins. Those are are precisely the types of queries that benefit the most from wco join algorithms. Various works have proposed or adapted wco joins to graph queries [2, 10, 13], with favorable results. Nevertheless, just for graphs with arity 3, a complete database index supporting wco joins requires $3! = 6$ index orders (tries simulated over B+-trees), leading to a high level of redundancy and high space requirements. To our knowledge, all wco algorithms up to date were mainly focusing on time performance rather than space usage, thereby ignoring a problem that becomes more pressing as larger datasets are handled.

To face the space problem, it has been recently proposed the *ring* [3], a read-only and main-memory succinct representation of triple datasets (e.g., RDF graphs), that regards the (subject, predicate, object) triples as cyclic and bidirectional, in a way that the 6 required orders can be represented using only one. As a consequence, the ring uses only sublinear additional space on top of the raw data. By storing the Burrows-Wheeler Transform (BWT) [7] of the set of triples instead of the raw triples, and by representing that BWT using wavelet trees [9], the ring enables worst-case optimal evaluation of multi-joins in compact space, with the Leapfrog Triejoin variable elimination algorithm [14].

Experimental results on the Wikidata graph [2] demonstrate that the ring uses 6% additional space beyond the raw data and 4–11 times less space than various prominent alternatives (EmptyHeaded, Jena, RDF-3X, Blazegraph, Virtuoso), being at the same time about 2–36 times faster in general to solve basic graph patterns. Overall, the ring offers the best performance and stability in query times while using a small fraction of the space required by several state-of-the-art approaches [3].

Despite these promising initial results, wco join algorithms have some shortcomings. First, even though the worst-case optimality is guaranteed regardless of the variable elimination order [12], it is known that some of them are much better than others [3]. Therefore, the biggest optimization problem is to implement multi-joins efficiently, because choosing the right order impacts the performance heavily. Second, it ignores binary join plans altogether, which have been shown to be efficient on many queries by decades of research in the field. In fact, the experiments showed that a wco-only join plan is not efficient for all types of queries [4]. Furthermore, since the 80s the scientific community knows that acyclic queries can be solved in instance-optimal runtime using Yannakakis algorithm [15], thus, solving acyclic queries in worst-case optimal time seems excessive. To face the latter problem, hybrid join plans have been proposed by various other works [1, 8, 11].

Therefore we think it is possible to improve the performance our Ring-based multi-join plan, by studying beyond worst-case optimality techniques, such as hybrid / adaptive plans, advance variable elimination order, and many others.

Additionally, we want to extend the ring to higher dimensions, such as relational databases with $d$ columns. Although a single ring cannot cover all the $d!$ orders needed to support worst-case optimality for $d > 3$, it has been proven that the number of orders needed to be stored is $O(2^d)$[3], and an order of magnitude lower than other alternatives[3] in practice. Moreover, we know that the ring is a special case to what has to been dubbed order graphs, which enables further reductions.

Lastly, we will study if an on-the-fly ring construction, and hybrid join plans can potentially produce better runtimes. The former alternative has the benefit of knowing the required order

[1]https://www.w3.org/RDF/
[2]https://www.wikidata.org/wiki/Wikidata:Database_download

during query evaluation, and the latter, inspired by a previous work [1], could allow us to handle more query patterns efficiently.

## 2 RESEARCH GOALS

As it has been demonstrated by previous works, the ring provides unmatched results in terms of space and runtime using a graph-adapted LTJ algorithm. The goal of this thesis is to expand the ring to support beyond worst-case optimality, building on in the state-of-the-art theoretical and practical results. To achieve this, we will exploit features inherent to the ring that are given by the wavelet tree structures that comprises it. For example, counting the number of different children for a given range can be calculated on wavelet trees.

- Graph Query Optimizations: Improving the graph-join wco algorithms, including techniques such as, hybrid join plans, and a better use of query structure and the database.
- Extending the Ring to Relational Databases: Adding support for higher arities ($d > 3$).
- Optional Extensions: Generalizing the supported language and collect improvements made to the ring in other works.

The first two objectives are the core of our plan and the third one collects a less clear and optional objective.

## 3 HYPOTHESIS

Adding hybrid plans that combine wco with non-wco joins, choosing a better variable elimination order by leveraging in-ring statistical information, and creating the ring indexes on-the-fly will provide a more competitive compact structure than its current implementation and than any of the classical RDBMSs indexes.

## 4 METHODOLOGY AND WORK PLAN

### 4.1 Methodology

We have established several goals and a hypothesis that represent our work plan. We will have frequent meetings and constant communication to discuss new ideas and results. Additionally, we will study the literature and propose new algorithms and data structures, which will be analyzed theoretically when possible and empirically in any case, implementing and comparing them against the state-of-the-art in established benchmarks.

### 4.2 Work Plan

The work plan is derived from the research goals. Although some of the tasks may overlap we will first focus on the graph query optimizations and second, in extending the ring to relational databases. The third objective is optional, we will evaluate pursuing it depending on the progress on the others.

As any research, we are expecting shifts in the plan, to which we will adapt accordingly.

#### *4.2.1 Graph Query Optimizations.*

- On-the-fly ring creation. This allows us to exclusively create and store the orders required to perform the query evaluation. First, each participating column of the join is represented a wavelet tree and the rest of the columns as a regular table. Then, multi-joins are executed using LTJ as usual.
- Design and implement ring-based binary joins. This corresponds to a traditional merge-join over wavelet trees. An initial approach is to measure how the algorithm behaves in a column

store, where there is one column per attribute and a single order. Then, we can compare the results with our multi-join algorithm.

- Study, implement and experiment the EmptyHeaded hybrid join plan with our ring-based LTJ[1]. This includes studying about generalized hypertree decompositions, Yannakakis algorithm and using ring-based binary joins.
- Design new hybrid plans that benefit from the ring structure.
- Study and design optimizations to choose a better variable elimination order, based on statistical information retrieved from the wavelet tree. An initial idea is to estimate the number of results of a join, to decide not only the variable order, but also if a binary join is more suitable than a multi-join, or vice-versa.

#### 4.2.2 *Extending the Ring to Relational Databases.*

- Study, design and implement a stand-alone version of a multidimensional ring based on the more general data organization idea of order graph [3].
- Investigate regarding high-arity vs. low-arity relations trade-off. Provide a theoretical analysis, Design, implementation, and experiment of new techniques on the ring to handle relations of variable arities.
- Study, analyze and design an ecosystem of indexes which generalize a column store data organization. This involves indexing columns on-the-fly to evaluate queries, caching of results, and a robust index pooling, among others.

#### 4.2.3 *Optional Extensions.*

- Generalize the supported language, which includes studying and implementing some other relational algebra operations, such has aggregations and projections.
- Integrate existing ring-based regular path queries implementation to our wco join algorithm [5].
- Extend the ring to support updates.

## REFERENCES

[1] Christopher R. Aberger, Andrew Lamb, Susan Tu, Andres Nötzli, Kunle Olukotun, and Christopher Ré. 2017. EmptyHeaded. *ACM Transactions on Database Systems (TODS)* 42 (2017), 1 – 44.

[2] Christopher R. Aberger, Susan Tu, Kunle Olukotun, and Christopher Ré. 2016. EmptyHeaded: A Relational Engine for Graph Processing. In *Proc. International Conference on Management of Data (SIGMOD).* 431–446. https://doi.org/10.1145/2882903.2915213

[3] 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 *Proc. International Conference on Management of Data (SIGMOD).* 102–114. https://doi.org/10.1145/3448016.3457256

[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* (Virtual Event, China) *(SIGMOD/PODS '21).* Association for Computing Machinery, New York, NY, USA, 102–114. https://doi.org/10.1145/3448016.3457256

[5] D. Arroyuelo, A. Hogan, G. Navarro, and J. Rojas-Ledesma. 2022. Time- and Space-Efficient Regular Path Queries. In *Proc. 38th IEEE International Conference on Data Engineering (ICDE).* To appear.

[6] A. Atserias, M. Grohe, and D. Marx. 2013. Size bounds and query plans for relational joins. *SIAM J. Comput.* 42, 4 (2013), 1737–1767. https://doi.org/10.1137/110859440

[7] Michael Burrows and David J. Wheeler. 1994. *A block-sorting lossless data compression algorithm.* Technical Report 124. Digital Equipment Corporation.

[8] Michael J. Freitag, Maximilian Bandle, Tobias Schmidt, Alfons Kemper, and Thomas Neumann. 2020. Combining Worst-Case Optimal and TraditionalBinary Join Processing.

[9] Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. 2003. High-order entropy-compressed text indexes. In *Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA).* 841–850.

[10] Aidan Hogan, Cristian Riveros, Carlos Rojas, and Adrián Soto. 2019. A Worst-Case Optimal Join Algorithm for SPARQL. In *ISWC (1)*. 258–275. https://doi.org/10.1007/978-3-030-30793-6_15

[11] Amine Mhedhbi and Semih Salihoglu. 2019. Optimizing Subgraph Queries by Combining Binary and Worst-Case Optimal Joins. *ArXiv* abs/1903.02076 (2019).

[12] Hung Q. Ngo, Christopher Ré, and Atri Rudra. 2013. Skew Strikes Back: New Developments in the Theory of Join Algorithms.

[13] Dung T. Nguyen, Molham Aref, Martin Bravenboer, George Kollias, Hung Q. Ngo, Christopher Ré, and Atri Rudra. 2015. Join Processing for Graph Patterns: An Old Dog with New Tricks. *In Proc. International Workshop on Graph Data Management Experiences and Systems* (2015). http://arxiv.org/abs/1503.04169

[14] Todd L. Veldhuizen. 2014. Leapfrog Triejoin: A Simple, Worst-Case Optimal Join Algorithm. In *Proc. 17th International Conference on Database Theory (ICDT)*. 96–106. https://doi.org/10.5441/002/icdt.2014.13

[15] Mihalis Yannakakis. 1981. Algorithms for Acyclic Database Schemes. In *Very Large Data Bases, 7th International Conference, September 9-11, 1981, Cannes, France, Proceedings*. IEEE Computer Society, 82–94.