UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

AUGMENTING LEAPFROG TRIEJOIN IN COMPACT GRAPH DATABASES TO SOLVE CONJUNCTIVE REGULAR PATH QUERIES

PROPUESTA DE TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS, MENCIÓN COMPUTACIÓN

DIEGO ARIAS

PROFESOR GUÍA:
GONZALO NAVARRO

SANTIAGO DE CHILE
2026

1. Introduction

Graph databases are an alternative tool for representing data. In contrast with traditional relational databases, which have multiple tables with different columns, graph databases are labeled graphs where nodes are subjects and objects, and edges are triples (s, p, o) connecting a subject s to an object o, with the predicate p serving as the edge label. Thus, a graph database can be represented as a set of triples (s, p, o).

Solving basic graph pattern (BGP) queries is an important problem in graph databases. A BGP is a set of triples (x, y, z) where each x, y, z is either a variable or a constant, forming a query graph. Any valid assignment to variables of the query graph such that the result is a subgraph of the data graph is a valid solution.

Obtaining solutions for a given BGP can be done by performing a join over the table (s, p, o), but common algorithms for solving this join might produce intermediate results that are larger than the final output. This has produced the need for algorithms that are worst-case optimal, that is, algorithms that solve the problem in time better than or equal to the worst-case output [1]. Many such algorithms exist, such as NPRR [2] and Leapfrog Triejoin (LTJ) [3], and they have been applied to graph databases in conjunction with compact data structures [4] in systems such as the Ring [5] and Compact Leapfrog Triejoin (CLTJ) [6]. These systems are capable of solving BGP queries in worst-case optimal time using very little space.

Regular path queries (RPQs) are restrictions of the form (x, R, z), where R is a regular expression and x and z are variables or constants. A solution satisfies such a restriction if there exists a path from x to z whose edge-label sequence matches R. RPQs generalize fixed-predicate BGP atoms: an ordinary triple pattern (x, p, z), where p is a fixed predicate, corresponds to the special case where R is the single label p. Finally, conjunctive regular path queries (CRPQs) combine these restrictions conjunctively, so a solution must satisfy all of them simultaneously. For our purposes, we consider CRPQs to also include BGP atoms directly, including atoms whose predicate is a variable.

Techniques from compact graph database systems like the Ring and CLTJ that were originally designed for solving BGPs have been applied to solving RPQs with surprisingly good results compared to the state-of-the-art, using the Ring [7], other compact graph representations [8] and compressed adjacency matrices [9]. But there is still no work on applying these techniques to achieve a complete system that can solve full CRPQs.

2. Related work

Ring

The Ring is a fairly recent compact graph database [5], which since its proposal has been iterated and improved upon [10]. It relies on the LTJ worst-case optimal join algorithm. Internally, it uses three wavelet matrices [11] to encode three orderings of (s, p, o) triples. This, together with some additional arrays, is sufficient to support the operations required by the LTJ algorithm in less space than the plain representation.

Compact Leapfrog Triejoin

CLTJ is a more recent compact graph database system that also uses the LTJ algorithm to solve BGP queries in graph databases [6]. It is implemented using compact tries for each (s, p, o) variable ordering. It also employs a technique called "trie switching" to avoid materializing all tries explicitly, further reducing space usage. Although it uses more space than the Ring, this results in better performance, as its basic operations are cheaper.

Regular Path Queries

The problem of solving RPQs in graphs is considerably harder than solving BGPs because the paths can be of arbitrary length and the number of paths between two nodes that satisfy a given regular expression can be exponential. Thus, for this work, we consider solutions to RPQs under set semantics, in other words the result is the set of (x, z) pairs such that a path exists between them that satisfies R.

Because graph databases are usually directional, two-way RPQs are RPQs where edges can be traversed in the other direction. From now on we make the convention that RPQs refer to these more general query types.

For our work, it is of particular interest to study how RPQs are solved using the Ring [7]. This is done by creating a Glushkov automaton [12] from the regular expression and then, for a fixed value of x or z, performing a BFS over the graph, extracting edges from the Ring representation while simulating the automaton in bit-parallel form. One caveat is that this approach doubles space usage by duplicating the graph and inserting the same edges with reversed predicates. This allows the algorithm to use backward search on the Ring to intersect active NFA nodes with outgoing graph edges in time proportional to the alternation complexity. For double-variable queries, they split the RPQs at mandatory positions, allowing them to start from a smaller candidate set if convenient.

Recent solutions that build on [7] use the same technique of traversing the product graph with bit parallelism, but rely on a different underlying graph representation [8]. They reduce space usage by avoiding the need to duplicate the graph, at the cost of trying each NFA node one by one. They also store arrays similar to those of the Ring, but do not represent them with wavelet matrices, leading to improved overall performance at the cost of worse performance in double-variable RPQs.

Finally RPQs can be reduced to boolean operations over the adjacency matrices of the graph, recent work has implemented these operations over sparse matrix representations [9] improving performance and space usage with respect to the Ring implementation.

Conjunctive Regular Path Queries

Recent theoretical work has shown that the usual notion of worst-case optimality for conjunctive queries does not transfer cleanly to CRPQs. In particular, for some CRPQs, matching the known worst-case output-size bounds would refute the Boolean matrix multiplication hypothesis considered in [13]. This suggests that worst-case optimality alone is not enough to guide practical CRPQ evaluation.

More recently, [14] introduced the first output-sensitive algorithm for acyclic CRPQs. Its running time depends on the input graph, the final query output, and a new parameter called contraction width, rather than on the potentially much larger outputs of the RPQ atoms. This improves over materialization-based approaches when the RPQs produce many pairs but the full CRPQ has few answers. At a high level, the algorithm reduces the query to free-connex acyclic pieces and evaluates them using calibrated RPQ outputs in a Yannakakis-style procedure [15].

There are many state-of-the-art implementations for solving CRPQs. Here we mention a few: relational DBMSs such as DuckDB [16] and Umbra [17], which use algebraic methods to evaluate RPQs and CRPQs; graph-specific systems such as Jena, Blazegraph [18] and Virtuoso [19], where Jena and Blazegraph uses navigational techniques over its graph representation, while Virtuoso implements a transitive closure operator over its relational engine; and more recent systems such as cuRPQ [20], which leverage GPU parallelism to achieve impressive performance using automata-based traversal techniques. Many other systems exist, but some do not support CRPQs, while others do not support the full feature set of RPQs.

3. Problem

Current techniques in compact graph databases have only been applied to solving single RPQs, of which 24% of real-world queries on Wikidata's public endpoint contain at least one. A system capable of solving C2RPQs with only two additional features (OPTIONAL and FILTER) could cover 46% of the query log and 76% of all unique queries [21].

Supporting CRPQs is a necessary step to achieving a complete database system that can handle more real-world query types while maintaining the primary benefit of compact graph database indices: considerably lower memory usage while retaining or improving performance.

A trivial solution to this problem would be to run all RPQs either at the start or at the end of the query. Our proposed approach entails modifying the LTJ algorithm to solve the

RPQ once some variable in it is instantiated during the execution of the join, computing, for a specific start or end node, the candidate values and incorporating the result into the LTJ intersection algorithm.

We propose working on three variants: one implemented over an instance of the Ring [10] applying the techniques from [8] to solve RPQs, one with two Rings, duplicating predicates as in the original Ring RPQ implementation [7], and one with CLTJ [6]. These variants have different limitations and capabilities, leading to novel and different approaches to handling multiple RPQs in a query.

Our proposed system would evaluate RPQs when they become connected to the bindings produced by the surrounding conjunctive query, so we do not expect to use the double-variable query optimizations from [7]. That case arises when both endpoints of an RPQ remain unbound, which in our setting means that the RPQ is disconnected, or not yet connected, to the rest of the query. This fully unbound case is not the main case of interest for CRPQs and is already covered by the original Ring RPQ work, while supporting it would increase space usage considerably.

With BGPs and worst-case optimal algorithms variable elimination orders (VEOs) are a very important factor for real world performance of these database systems [6], thus a hard problem of this approach is introducing RPQs nicely into the existing query planners, allowing for restrictive RPQs to be instantiated early while leaving the rest for later. A specific challenge is leveraging the extra functionality of the Ring compared with CLTJ to calculate query plans.

4. Research questions

- Does integrating RPQ solving during the execution of the LTJ algorithm improve execution times compared with doing it at the start or end?
- Does this approach using compact graph database systems to solve CRPQs perform better than complete state-of-the-art graph databases?
- Between the possible variants proposed, which performs best on real-world CRPQs?
- Which techniques for defining VEOs performs best on real-world queries?

5. Hypothesis

Our main hypothesis is that solving RPQs during the execution of the LTJ algorithm will perform better than alternative solutions that frontload it or leave it for the end. This is because we believe that evaluating RPQs at the start can lead to many useless intermediate results before even starting the rest of the query, whereas leaving it for the end means that RPQs that could have limited the search space were not leveraged.

We also expect all of these solutions to perform better than state-of-the-art non-compact graph database systems at solving CRPQs, as was the case for RPQs [7–9]. While

CRPQs introduce additional challenges because RPQ evaluation must be combined with join processing, the previous results for RPQs and BGPs suggest that compact graph representations should remain competitive in this setting.

Between the variants, we expect a version using two Rings to perform best, followed by the one Ring variant and then CLTJ. This is due to restrictions on the available operations in CLTJ with respect to the Ring, which limit the query planning phase, although we do expect CLTJ to perform better on some queries where traversal speed matters most. In terms of space usage we expect one Ring to use the least, followed by two Rings, then CLTJ.

6. Objectives

General objectives

Improve the state-of-the-art for solving CRPQs on graph databases by leveraging the benefits of compact data structures and worst-case optimal join algorithms.

Specific objectives

- Implement three variants of existing compact graph database systems that are capable of solving CRPQs in better than state-of-the-art performance.
- Benchmark the implemented solutions and compare with existing state-of-the-art systems.
- Understand which techniques and modifications of the LTJ algorithm are best at handling real-world queries.

7. Methodology

What follows is a proposed methodology with steps considered necessary for the completion of the work described.

1. Review of prior work in the area, specifically related to compact graph databases, solving RPQs on these systems and the state-of-the-art on solving CRPQs.
2. Define the limitations and capabilities of each possible variant, and decide based on this the approach to be used in each.
3. Design and implement the three proposed variants based on existing implementations.
4. Do testing across variants to ensure correctness.
5. Obtain query logs and filter for CRPQs to test the performance of the implementations.
6. Explore and decide which competing graph database systems to compare with.
7. Execute performance tests on the implemented solutions and other database systems.
8. Compare the results between variants and with other database systems.

We expect to use the Wikidata graph [22] and its real-world query logs [21] as the dataset we will test against, mainly for its scale.

Our work is expected to be built on top of existing implementations of the Ring and CLTJ, both written in C++, with the resulting implementation being available as open source software.

8. Expected results

The expected contribution of this work is a functioning implementation of three variants of existing compact graph database systems that are capable of solving CRPQs with better performance and lower space usage than the current state-of-the-art solutions.

We also expect to better understand which techniques work best when solving CRPQs, specifically what modifications to the LTJ algorithm and query planning stage lead to improvements for different query shapes.

Bibliography

[1] A. Atserias, M. Grohe, and D. Marx, "Size Bounds and Query Plans for Relational Joins," *SIAM Journal on Computing*, vol. 42, no. 4, pp. 1737–1767, 2013, doi: 10.1137/110859440.

[2] H. Q. Ngo, E. Porat, C. Ré, and A. Rudra, "Worst-case optimal join algorithms: [extended abstract]," in *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, in PODS '12. Scottsdale, Arizona, USA: Association for Computing Machinery, 2012, pp. 37–48. doi: 10.1145/2213556.2213565.

[3] T. L. Veldhuizen, "Leapfrog Triejoin: a worst-case optimal join algorithm," *CoRR*, 2012, [Online]. Available: http://arxiv.org/abs/1210.0481

[4] G. Navarro, *Compact Data Structures: A Practical Approach*. Cambridge University Press, 2016.

[5] D. Arroyuelo, A. Hogan, G. Navarro, J. L. Reutter, J. Rojas-Ledesma, and A. Soto, "Worst-Case Optimal Graph Joins in Almost No Space," in *Proceedings of the 2021 International Conference on Management of Data*, in SIGMOD '21. Virtual Event, China: Association for Computing Machinery, 2021, pp. 102–114. doi: 10.1145/3448016.3457256.

[6] D. Arroyuelo *et al.*, "CompactLTJ: Space & Time Efficient Leapfrog Triejoin on Graph Databases," *The Very Large Databases Journal*, vol. 34, p. article67, 2025.

[7] D. Arroyuelo, A. Gómez-Brandón, A. Hogan, G. Navarro, and J. Rojas-Ledesma, "Optimizing RPQs over a compact graph representation," *The VLDB Journal*, vol. 33, no. 2, pp. 349–374, Sep. 2023, doi: 10.1007/s00778-023-00811-2.

[8] G. Navarro and J. Robert, "Compressed Graph Representations for Evaluating Regular Path Queries," in *String Processing and Information Retrieval*, Z. Lipták, E. Moura, K. Figueroa, and R. Baeza-Yates, Eds., Cham: Springer Nature Switzerland, 2025, pp. 218–232.

[9] D. Arroyuelo, A. Gómez-Brandón, and G. Navarro, "Evaluating regular path queries on compressed adjacency matrices," *The VLDB Journal*, vol. 34, no. 1, Nov. 2024, doi: 10.1007/s00778-024-00885-6.

[10] D. Arroyuelo, F. Barisione, A. Fariña, A. Gómez-Brandón, and G. Navarro, "New compressed indices for multijoins on graph databasesV," *Information Systems*, vol. 137, p. 102647, 2026, doi: https://doi.org/10.1016/j.is.2025.102647.

[11] F. Claude and G. Navarro, "The Wavelet Matrix," in *String Processing and Information Retrieval*, L. Calderón-Benavides, C. González-Caro, E. Chávez, and N. Ziviani, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 167–179.

[12] V. M. Glushkov, "The abstract theory of automata," *Russ. Math. Surv.*, vol. 16, no. 5, pp. 1–53, Oct. 1961.

[13] T. Cucumides, J. Reutter, and D. Vrgoč, "Size Bounds and Algorithms for Conjunctive Regular Path Queries," in *26th International Conference on Database Theory (ICDT 2023)*, F. Geerts and B. Vandevoort, Eds., in Leibniz International Proceedings in Informatics (LIPIcs), vol. 255. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, pp. 1–17. doi: 10.4230/LIPIcs.ICDT.2023.13.

[14] M. A. Khamis, A.-M. Hurjui, A. Kara, D. Olteanu, D. Suciu, and Z. Tian, "Output-Sensitive Evaluation of Acyclic Conjunctive Regular Path Queries," in *29th International Conference on Database Theory, ICDT 2026, Tampere, Finland, March 24-27, 2026*, B. ten Cate and M. Funk, Eds., in LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2026, pp. 1–20. doi: 10.4230/LIPICS.ICDT.2026.18.

[15] M. Yannakakis, "Algorithms for acyclic database schemes," in *Proceedings of the Seventh International Conference on Very Large Data Bases - Volume 7*, in VLDB '81. Cannes, France: VLDB Endowment, 1981, pp. 82–94.

[16] M. Raasveldt and H. Mühleisen, "DuckDB: an Embeddable Analytical Database," in *Proceedings of the 2019 International Conference on Management of Data*, in SIGMOD '19. Amsterdam, Netherlands: Association for Computing Machinery, 2019, pp. 1981–1984. doi: 10.1145/3299869.3320212.

[17] T. Neumann and M. J. Freitag, "Umbra: A Disk-Based System with In-Memory Performance," in *Conference on Innovative Data Systems Research*, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:209379505

[18] B. B. Thompson, M. Personick, and M. Cutcher, "The Bigdata® RDF Graph Database," in *Linked Data Management*, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:33186182

[19] O. Erling, "Virtuoso, a Hybrid RDBMS/Graph Column Store.," *IEEE Data Engineering Bulletin*, vol. 35, no. 1, pp. 3–8, 2012.

[20] S. Park, S. Kim, and M.-S. Kim, "cuRPQ: A High-Performance GPU-Based Framework for Processing Regular and Conjunctive Regular Path Queries." [Online]. Available: https://arxiv.org/abs/2602.20748

[21] A. Bonifati, W. Martens, and T. Timm, "Navigating the Maze of Wikidata Query Logs," in *The World Wide Web Conference*, in WWW '19. San Francisco, CA, USA: Association for Computing Machinery, 2019, pp. 127–138. doi: 10.1145/3308558.3313472.

[22] D. Vrandečić and M. Krötzsch, "Wikidata: a free collaborative knowledgebase," *Commun. ACM*, vol. 57, no. 10, pp. 78–85, Sep. 2014, doi: 10.1145/2629489.