Revision Plan for paper “Towards a Converged Relational-Graph Optimization Framework”

Anonymous Authors

1 MODIFICATIONS TO MAKE

We sincerely appreciate the feedback and suggestions from the reviewers and the meta-reviewer. According to the comments[1], in this revision, we are going to make the following modifications, including:

- adding experiments
- providing additional related works
- adding examples
- adjusting descriptions based on importance
- polishing this paper

In terms of adding experiments, we plan to revise this paper from the following aspects:

- We will add Umbra as a new baseline and conduct comprehensive experiments on it. In detail, we plan to generate query plans optimized with Umbra’s optimizer on LDBC SNB and JOB benchmarks, and then hardcode the plans to run within DuckDB for a fair comparison. (R2-i.4, R2-iii.1, R2-C1, R2-C3, R3-O2.3, R3-C2, M-C1, M-C3). Please note that to the best of our knowledge, Umbra is not open-sourced. We have sent an email to the authors of Umbra requesting the source code, but we are not sure whether we will be able to obtain it and have enough time to conduct experiments with it. Therefore, if we cannot obtain the code, we may conduct experiments on another baseline, i.e., EmptyHeaded[2] or LevelHeaded[3] (if available), which also support worst-case optimal joins. In that case, Umbra in this revision plan will be replaced with the new baseline.
- We will conduct comprehensive experiments on a larger instance, i.e., LDBC G_{sf100} to make the conclusions more convincing. (R3-O2.6, R3-C2)
- We will conduct some more experiments on Calcite[4] to show the importance of reducing the time cost of query optimization. (R2-S1, R2-ii.1, M-C6).

In terms of providing additional related works, we plan to modify this paper from the following aspects:

- We will discuss the differences between tree decomposition applied in this paper and the techniques used by EmptyHeaded, CTJ respectively. (R2-i.1, R2-C2, M-C2)
- We will explain the relationships between DuckPGQ and DuckDB in more detail in this paper. (R2-i.2, R2-C2, M-C2)

In terms of adding examples, we plan to modify this paper from the following aspects:

- We will add a case study in the revision to demonstrate why plans generated by RelGo are better than those generated by baselines. (R1-O2, M-C4)
- We will add an example to show what is given up when using the tree decomposition approach in RelGo. (R2-ii.1, M-C4)
- We will move some representative SQL/PGQ queries used in the experiments to the text. (R3-O2.1, M-C4)
- We will add an example of a converged query plan in the revision. (R3-O2.4, M-C4)

In terms of adjusting descriptions based on importance, we plan to revise this paper from the following aspects:

- We will introduce the higher-order graph statistics in more detail. (R1-O1, M-C5)
- We will explain more about global and local optimization rules applied in RelGo and explain their potential tradeoffs. (R2-i.3, M-C6)
- We will add some statements to emphasize that the intersection evaluation using worst-case optimal join is implemented on relational tables. (R2-i.5)
- We will simplify the contents about the definition of RGMapping, simplify the needed notation, remove RGMapping from the contributions of this paper, and express some concepts more clearly in E-R terms. (R2-i.6, R3-O1.1, R3-M1, R3-C1, M-C7)
- We will include the versions of DuckDB used in the experiments. (R2-iii.2)
- We will introduce the procedures for data loading and graph index construction in this paper in more detail. (R3-O2.2)
- We will explain some confusing statements in this paper more carefully. (R3-O2.5)
- We will discuss about how to extend RelGo to deal with queries without explicit PGQ component. (R3-O3.1, R3-C3)
- We plan to explain why we limit the extensive depth, methods, and literature on relational query optimization in this revision and add more descriptions about relational query optimizations. (R3-O4.1, R3-C3)

Finally, in terms of polishing this paper, we plan to make the following modifications:

- We will polish figures and enlarge the font sizes of some figures. (R1-D1, R1-M2)
- We will fix typos in the revision. (R1-M1)
- We will move Table 1 earlier and make the query example on Page 1 a figure float. (R3-M1)

Detailed responses to reviewers and the meta-reviewer are as follows.

[1]In this section, “Ri-xxx” represents a comment from Reviewer #i and “M-xxx” represents a comment from the meta-reviewer.

[2]Christopher R. Aberger, Susan Tu, Kunle Olukotun, Christopher Ré: EmptyHeaded: A Relational Engine for Graph Processing. SIGMOD Conference 2016: 431-446.

[3]Christopher R. Aberger, Andrew Lamb, Kunle Olukotun, Christopher Ré: LevelHeaded: A Unified Engine for Business Intelligence and Linear Algebra Querying. ICDE 2018: 449-460.

[4]https://calcite.apache.org/

2 DETAILED RESPONSE TO THE FEEDBACKS

2.1 Response to Reviewer #1

O1. I did not understand from the discussion in section 4.2 what are all types of statistics that are maintained to support the optimization. Does the converged approach work with common types of statistics used on relational data or some adjustments were necessary? Did GLogue data structure also require adjustments compared to the graph native design? I would appreciate more details in this area of the design, especially around the higher-order graph statistics.

Response: Thank you for your suggestion! We will introduce the higher-order graph statistics in more detail and explain the relationship between the statistics used in this work and those used in relational databases and graph databases in the revision.

O2. It is not clear to me what are typical shapes of the plans that lead to such speedups. It would be nice to include representative examples of more efficient plans.

Response: Thanks for your suggestion! We will add an example in the revision, which includes the execution plans obtained by our converged optimizer and the baseline optimizers, respectively, to explain the reason for the efficiency of our method.

D1. Many aspects of the presentation of figures can be improved: (1) Workflow illustrations on figures 1,2 and 5 have tiny graphs (2) Figures 6–10 showing experimental results use tiny fonts.

Response: Thanks for your suggestion ! We will polish the figures in the revision and enlarge the font sizes.

M1. As DRAM nowadays comes in sizes that are multiple of 32GB, it is quite unusual to have 251GB. Is this a typo?

Response: Thanks for your suggestion! We will double check the size of the DRAM and report the correct number in the revision.

M2. Please consider normalizing all runtimes to one system, perhaps DuckDB, and plotting speedups achieved by the other two in figure 10.

Response: Thanks for your suggestion! We will redraw Figure 10 and normalize the time cost as you have advised.

2.2 Response to Reviewer #2

S1. However given how read-heavy the queries are, I wonder if the query optimization time matters in practice.

Response: Thanks for your suggestion! We will conduct some experiments to show that some commonly used optimizers in practice (e.g., Calcite) may cost a long time for query optimization, which shows the importance of reducing the time of query optimization.

i.1. I think it is important to mention the differences that exist between the techniques used by EmptyHeaded [r3] and CTJ [r4] and tree decomposition in this manuscript.

Response: Thanks for your comment! We will discuss about the differences between these techniques and RelGo proposed in this paper in more detail in the revision.

i.2. How do you contrast your approach with the query evaluation where the DuckDB team introduces SQL/PGQ through the extension framework in [40] and [41].

Response: Thank you for your comment! As stated in paper [40], "DuckPGQ translates SQL/PGQ queries into a pure SQL query plan which gets executed by DuckDB as a normal query (with some UDF calls)". Therefore, for LDBC SNB IC queries and JOB queries without UDF calls, the optimizer of DuckPGQ effectively becomes that of DuckDB, which is already tested in this paper. In the revision, we will explain the relationships between DuckPGQ and DuckDB.

i.3. filter push-down, called 'FilterIntoMatchRule' is an artifact of separating the evaluation into a matching operator instead of a global optimization approach and is seen as necessity in my opinion and not a new optimization technique.

Response: Thank you for your comment! FilterIntoMatchRule is indeed necessary in practice, which has also been proved by our experimenal results in Section 5.2. Besides, such a rule can only be applied due to our design of the converged optimization framework.

We will talk more about the local and global optimization rules applied in RelGo and explain their potential tradeoffs in more detail.

i.4. We need a comparison to Umbra and perhaps making it apple-to-apple means obtaining the plan from Umbra and hardcoding it to run within your DuckDB implementation.

Response: Thank you for you suggestion! We will compare Umbra with RelGo in the revision. In detail, we will first generate query plans on LDBC SNB and JOB benchmarks optimized with Umbra's optimizer. Then, we will hardcode the plans to run within DuckDB for a fair comparison.

i.5. The intersection evaluation using WCOJ is very similar to that in [34] and it is hard to tell the difference. My understanding is there is no relational tables in that case.

Response: Thank you for your comment! It seems that citation [34] is the standard of SQL/PGQ and wcoj is not mentioned in that standard. Perhaps you are talking about citation [43] in the paper, i.e., Zhengyi Yang, Longbin Lai, Xuemin Lin, Kongzhang Hao, and Wenjie Zhang. 2021. HUGE: An Efficient and Scalable Subgraph Enumeration System.

Then, your are correct that the intersection evaluation using worst-case optimal (wco) join in [43] is for graphs and that in this paper is implemented on relational tables. We will add some statements to emphasize this difference in the revision.

i.6. Most of the mapping work is assumed based on the SQL spec and is in use already as per [40] and [41] which is why I do not see it as a contribution. Cypher-based systems like Kuzu do this mapping in their DDL as well and are able to query from relational stores directly.

Response: Thanks for your comment! Actually, we use the mapping in this paper to formally define the relationships between relational tables and graph vertices/edges. We agree with you that SQL/PGQ has defined the mapping in a grammar manner and DuckPGQ/Kuzu have implemented such grammars in their implementations. In this revision, we will remove the contents about RGMapping from the contributions listed in Section 1.

ii.1 verifying that truly this notion of graph-aware optimization is what lead to the gains requires a clear discussion of i) and further highlighting important design goals that make the smaller search space aspect appealing. It is unclear what is given up when using the tree decomposition approach proposed.

Response: Thanks for your suggestion! In this revision, we will compare the time cost of query optimization using Calcite and RelGo to demonstrate the importance of accelerating the query optimization process. Besides, we will give an example to show what is given up when using the tree decomposition approach.

iii.1 using DuckDB as a baseline can be misleading in the results as DuckDB doesn't not contain a state of the art query optimizer, the focus of this paper, and does not support worst-case optimal joins.

Response: Thanks for your suggestion. We will add Umbra as a new baseline, which supports worst-case optimal joins.

iii.2 DuckDB and GrainDB are possibly of different versions and so is the authors implementation. This level of detail is not reflected in the experimental section.

Response: Thank you for your comment! In the experiments, to ensure a fair comparison, we have implemented GrainDB on DuckDB v0.9.2, which is the version of DuckDB used in other experiments. We will add the statements in the revision.

C1. Generate Umbra plans, hardcode these plans within the authors solution using their operators and run the plan (multi-way joins) should use the adj. list intersection. Compare the plan with that generated by your system.

Response: Thanks for you suggestion! In the revision, we will generate Umbra plans and conduct experiments on LDBC SNB and JOB benchmarks. The results will be compared with RelGo.

C2. Add some of the work that I mention as missing to the related work section.

Response: Thanks for your suggestion! We will add the related works you mentioned to the related work section in the revision. Besides, we will explain the difference between our tree decomposition in RelGo with that of EmptyHeaded in more detail.

C3. Getting rough numbers from Umbra and Kuzu as sanity checks would be nice to have but not necessary.

Response: Thanks for your suggestion. We will add the experiments about Umbra in this revision. Moreover, since Kuzu is a graph database, which is not our focus in this paper, it may be not suitable to be used as a baseline in this paper. Therefore, we will add it as a related work.

2.3 Response to Reviewer #3

O1.1 SQL/PGQ presented in an obtusely formal way that adds little to the paper or the readers' understanding, but rather works against it. Meanwhile, the explication misses some seemingly rather important points. For instance, zeta for vertex and edge mappings? These are introduced, but then never used again. But really, the authors spend much realestate to formalize the SQL/PGM mapping, when that is not a contribution of the paper. What should be introduced here should just be enough that the casual but knowledgeable reader should understand the concept. There is no guaranteed bijection here. The relational model does not use object IDs; graph data models do. The introduction of UUIDs skirts the issue, to be replaced shortly later in the paper with row-IDs. It seems rather that one would want to insist that any table manifested as an labeled edge set via GRAPH_TABLE have foreign keys to both the node sets / entity tables it is being used to bridge. This could be conveyed more simply perhaps in E-R terms.

Response: Thank you for your comment! We will simplify the formal definition of RGMapping for better understanding of this paper and remove the notations that never used in the paper. Besides, we will express these concepts more clearly in E-R terms.

O2.1 No concrete insight is offered into the test queries, nor concrete examples.

Response: Thanks for you suggestion! We will move some representative SQL/PGQ queries used in this paper to the Evaluation section and explain how these queries are generated.

O2.2 Not everyone will be familiar with [21]. The paper needs to be self-contained.

Response: Thank you for your suggestion! We will introduce the procedures for data loading and graph index construction in this paper in more detail to make this paper more self-contained.

O2.3 While DuckDB has become quite popular, I doubt many would argue that it sets the gold standard for relational query optimization. I recommend that at least an additional relational engine be included, perhaps PostgreSQL, in the comparisons. The authors extended a front-end to DuckDB to handle SQL/PGQ queries, although I do not understand why. Poor translation of the queries into plain SQL — DuckDB is used to represent the graph-agnostic approach — could be what hinders its performance. The reader presently cannot discern. Instead, having pairs in the test suite of SQL and SQL/PGO that are semantically equivalent but manually composed I believe would be fairer. And these could be used, say, for PostgreSQL without needing to reinstrument it.

Response: Thanks for your suggestion! Since other reviewers and the meta reviewer strongly recommend that we add Umbra as a new baseline, we will conduct more experiments on Umbra to demonstrate the superiority of our converged optimizer.

Additionally, we use the engine of DuckDB in this paper because one of our baselines, i.e., GrainDB, is implemented on it, and it would require a lot of extra effort to use another engine. Moreover, we are planning to implement RelGo as an embedded optimizer in the future so that different relational engines can leverage the capabilities of RelGo with minimal effort.

O2.4 A concrete example of a "converged" query plan would be exceedingly helpful.

Response: Thanks for your suggestion! We will add an example of a converged query plan in the revision.

O2.5 And a less simplistic discussion of the plans and operators. For example, at "In the absence of a graph index, HASH_JOIN is used for the entire plan, where the cost is simply the product of the cardinalities of the two relations being joined," I lost the narrative somewhat.

Response: Thanks for your suggestion! We agree that this sentence is a bit confusing. In fact, it means that when we are dealing with the matching operators, the joins obtained by decompostion are implemented as HASH_JOINs. As these joins connect vertices to their adjacent edges, it is always far from cross product. We will explain it in more detail in the revision to avoid confusion.

O2.6 The paper could be strengthened by adding significantly larger data instances.

Response: Thanks for your suggestion! We will conduct comprehensive experiments on a much larger data instance $G_{sf_{100}}$, with scale factor 100, generated by the official LDBC Data Generator.

O3.1 SQL/PGQ goes against the declarative ideal of "state what you want, not how to do it." Could RelGo be extended to additionally rewrite SQL/PGQ queries, and SQL queries with no explicit PGO component in some cases, to consider "SPJR" query plans? Providing the discussion and a roadmap for doing so would greatly strengthen the contributions.

Response: Thank you for your suggestion! It is an interesting idea and we will add some discussion about this.

from our perspective, it is possible to extend RelGo to consider "SPJR" query plans. First of all, the optimizer should know how to convert relational tables to vertices and edges in graphs. A intuitive way is provided by Boudaoud[1]. In practice, whether a relational table can be converted to edges is determined by the existence of graph indexes. Without the existence of corresponding graph indexes, a relational table can never be converted to edges.

Given a SQL query, by transforming arbitrary relational tables into vertices or edges, the original SQL query is converted into a SQL/PGQ-like query with several matching operators, where each matching operator contains a connected pattern graph. The results of the matching operators are joined together to get the final results. Since the cost of matching operators and relaional operators are computable, by traversing different methods to transforming relational tables into vertices and edges, we can find the best execution plan. However, the problem of this method lies in the vast search space and it needs further study.

O4.1 The authors short the extensive depth, methods, and literature on relational query optimization. While the authors are constrained by space and focus by the conference paper format, explanation of how and why they limit their focus in the ways they do should be offered.

Response: Thanks for your suggestion! We short the descriptions of relational query optimizations because any relational optimizer can be plugged into RelGo. In the revision, we will add more descriptions of relational query optimizations.

M1. I would recommend moving Table 1 earlier; and also simplifying the needed notation. I think the query example on page 1 would be better as a figure float.

Response: Thank you for your suggestion! We will move Table 1 earlier and simplify the needed notation in the revision. Moreover, we will make the query example on Page 1 a figure float.

C1. My O1 and O2. The paper does not convey enough in concrete terms how their approach works overall that the reader can take concrete knowledge away from it. The abundance of the formal discussions of SQL/PGO can be traded to provide more in depth, concrete insight into RelGo.

Response: Thanks for your suggestion! We will simplify the formal discussions of SQL/PGQ and focus more on our contributions.

C2. I think additional experiments are needed, as I outline in O2. If not, then the authors should make a compelling case in discussion in the paper as to why they are not needed.

Response: Thank you for your suggestions! We will add more experiments with one more baseline (Umbra) and a larger instance (G_{sf100}) in the revision.

[1]Boudaoud A, Mahfoud H, Chikh A. Towards a Complete Direct Mapping from Relational Databases to Property Graphs. International Conference on Model and Data Engineering 2022: 222-235.

C3. Addressing O3 & O4 to some extent would strengthen the paper, I believe.

Response: Thank you for your comment! We will try our vest to address O3 & O4 as described above.

2.4 Response to the Meta Reviewer

C1. Comparison with prior work in particular Umbra emerged as an important consideration among other required improvements listed below.

Response: Thanks for your suggestion! We will add Umbra as a new baseline and compare it with our converged optimizer.

C2. Significant relevant related work is not discussed, compared, and contrasted. [R2] a. Add reference, discussion, and comparison with [r1–r5] from R2a-iii.

Response: Thanks for your comment! In the revision, we will add the references to [r1–r5] from R2a-iii and make some discussion and comparison w.r.t. these studies.

C3. b. Add experimental evaluation comparing against Umbra and Kùzu. Redo the experiments with larger instances.

Response: Thank you for your suggestion! We will add Umbra as a new baseline in this revision and conduct experiments on both LDBC SNB and JOB benchmarks. However, since Kùzu is in fact a graph database, which is not our focus in this paper, it may be not suitable to be used as a baseline in this paper. We will add Kùzu as a related work and discuss about it in the revision.

Besides, we will redo the comprehensive experiments on a larger instance, i.e., G_{sf100} with scale factor 100, generated by the official LDBC Data Generator.

C4. 2. More concrete discussion and explanation about the optimization platform and in the experiments.
a. Add concrete examples of queries and query plans used in the experiments.

Response: Thanks for your suggestion! In this revision, we will add a case study to demonstrate why plans generated by RelGo are better. Besides, we plan to add examples to show what is given up when using the tree decomposition approach in RelGo. Moreover, we will move some representative queries used in the experiments to the text. Furthermore, we will add a converged query plan.

C5. b. State clearly "all types of statistics that are maintained to support the optimization." [R1-O1]

Response: Thanks for your suggestion! We will explain the higher-order graph statistics used in RelGo in more detail.

C6. c. Be much clearer in discussion about the tradeoffs being made here between local and global optimization. And that involved in restricting to a smaller search space. Show that the optimization time is relevant in practice.

Response: Thank you for your suggestion! We will explain more about the global and local optimizations applied in RelGo. Besides, we will conduct some experiments on Calcite to show the importance of reducing the time cost of query optimization.

C7. 3. Rewrite the presentation of SQL/PL. [R2-i; R3-O1]

Response: Thank you for your suggestion! We plan to simplify the contents about the definition of RGMapping and simplify the needed notations in the revision.