View Reviews

Paper ID
56

Paper Title
TreeTracker Join: Turning the Tide When a Tuple Fails to Join

Track Name
Round 1 Paper Submissions

Reviewer #2

Questions

1. Overall evaluation
Accept with Shepherding

3. Novelty
Novel solution

4. Importance: select all that apply:
SIGMOD attendees will learn something interesting from the paper

5. Summary of contribution (in a few sentences)

The paper introduces a new join algorithm, the TreeTracker Join (TTJ),
which is a direct extension of the TreeTracker algorithm,
a constraint satisfaction problem (CSP) algorithm.
TTJ merges two search techniques, backjumping and no-good, which are applied during join evaluation
to remove a minimum number of join-fail tuples.
The new TTJ is based on the observation that query evaluation can be treated as a CSP problem.
TTJ is compared with several existing algorithms, including a classical hash-join,
a classical semijoin method (Yannakakis's algorithm), the Predicate Transfer algorithm,
and the Lookahead Information Passing algorithm.
Theoretical discussions, complexity analysis, and an extensive experimental section provide strong arguments
that the TTJ algorithm is superior to the aforementioned algorithms.

6. Describe in detail all strong points, labeled S1, S2, S3, etc.

S1. The TTJ algorithm is described in great detail, with a comprehensive set of working examples illustrating how the algorithm functions.
S2. The TTJ algorithm is compared with the classical Hash Join (HJ) algorithm, Predicate Transfer (PT) algorithm,
Lookahead Information Passing (LIP) algorithm, and Yannakakis's (YA) algorithm.
The paper describes and contrasts these algorithms with TTJ.
S3. The experimental section is extensive, utilizing new, uniform implementations of the algorithms.
The data sets and query workloads cover a diverse set of scenarios.
S4. Section 7, "Limitations and Future Work" clearly outlines the limitations of the TTJ algorithm.
The authors cite an anonymous follow-up paper which addresses some of these limitations.

7. Describe in detail all opportunities for improvement, labeled O1, O2, O3, etc.

O1. O1 is similar to S4 above:
Section 7, "Limitations and Future Work" clearly outlines the limitations of the TTJ algorithm.
The authors cite an anonymous follow-up paper which addresses some of these limitations.

It is encouraging that the authors have a followup paper. However, we cannot read this paper to
find out how the limitations are addressed.

O2. Some figures are hard to read - see 'Minor remarks' below for some ideas of how to improve readability.

O3. The experimental section should include edge scenarios where no tuples qualified to be removed by TTJ but there are tuples
which are removed by other algorithms. How does TTJ perform in these scenarios?

O4. The experimental section should include edge scenarios where no tuples qualified to be removed by any of the algorithms.

How does TTJ perform in these scenarios?

8. Please rank the three most critical strong points and/or opportunities for improvement that led to your overall evaluation rating, e.g., "{S1, O7, O4}"
S1, S2, S3, O3, O4, S4, O2

9. Inclusive writing: does the paper follow the SIGMOD inclusive writing guidelines? See https://dbdni.github.io/pages/inclusivewriting.html for more information. If not, please provide suggestions on how to improve the writing of the paper.
I agree

10. D&I compliance
Yes

11. Availability: We have asked authors of all papers to make their code, data, scripts, and notebooks available, if this is possible. You can find this information in the summary of the paper on CMT when clicking on the Paper ID next to the title. In the text box provided below, please provide (a) constructive feedback to the authors about these provided artifacts (if any), (b) how this was useful in your evaluation of the submission, and (c) what changes are needed in order to make the code/data/scripts/notebooks shareable with the community (if any).
The code is provided and it is readable. I did not run the code.

12. Minor remarks. Use this field to describe minor remarks that are not listed as opportunities for improvement, e.g., to highlight typos, formatting problems, or minor technical issues.

Figure 7 is very hard to read due to the skinny bars - please adjust to improve readability.
In all Figures, green colour is too close to blue colour hence hard to distinguish - please choose a different colour for one of them - e.g., replace blue with mauve.

13. List required changes for a revision, if appropriate, by identifying subset of previously specified opportunities for improvement (e.g., O1, O3, O6).
None

Reviewer #3

Questions

1. Overall evaluation

Accept with Shepherding

3. Novelty

Novel solution

4. Importance: select all that apply:

The paper is likely to influence other research in the community

5. Summary of contribution (in a few sentences)

The paper presents TreeTracker Join (TTJ), a hash-based join algorithm, that has the nice property that intermediate result tuples are non-dangling without having to eliminate all dangling tuples from the join operands. TTJ is based on an extension of a CSP algorithm that uses two search techniques (backjumping and no-good) to eliminate some dangling tuples from the join operands. An experimental evaluationcomparing four baseline techniques demonstrate the potential of TTJ.

6. Describe in detail all strong points, labeled S1, S2, S3, etc.

S1. TTJ provides the optimality guarantee (with non-dangling intermediate result tuples) without the need to eliminate all dangling tuples in the join operands.

S2. While the idea of formulating query evaluation as a constraint satisfaction problem is not new, this paper presents an interesting attempt to revive this research direction.

S3. The algorithms are explained with useful examples.

7. Describe in detail all opportunities for improvement, labeled O1, O2, O3, etc.

O1) A key concern with the paper is the experimental evaluation which does not convincingly demonstrate the effectiveness of TTJ. Specifically, several experimental design decisions are stated without any justifications.
(a) Why are the experiments conducted in the context of federated database systems using a single machine?
(b) Why are the experiments restricted to using only one CPU core? How do the techniques compare without this restriction?
(c) Given the 64GB of RAM and the scale factor of 1 used in both TPC-H and Star Schema benchmarks, is the entire database and all the hash tables in the pipelined evaluation resident in main memory? That is, the reported running times measure only CPU processing (ie without any disk I/O)? How significant is the improvement of TTJ over competing techniques if the join processing is dominated by disk I/O time?
(d) Given that WCOJ algorithms share the same goal as TTJ to produce non-dangling intermediate join results, why are WCOJ algorithms not included as relevant baselines?
(e) Since TTJ has been extended to process cyclic queries, why are the experimental queries restricted to acyclic queries?

O2) Additional comments on experimental evaluation.

(a) If the implementation of TTJ (and all the baseline techniques) assumes that the data are main-memory resident, it would be insightful to include some main-memory database system for comparison as well.

(b) How significant is the contribution of no good technique to TTJ performance? It would be interesting to include a variant of TTJ that uses only backjumping without no good for comparison.

O3) With regards to the first contribution stated in Section 1 on TTJ's use of CSP techniques, is this novel given TreeTracker [10] and the follow-up paper: Daniel P. Miranker, Roberto J. Bayardo Jr., Vasilis Samoladas, "Query Evaluation as Constraint Search; An Overview of Early Results", CDB 1997: 53-63.

O4) With regards to the second contribution on the construction of join tree from query plan and vice-versa, it's not clear to me why it's important to have bi-directional contruction. Section 3.1 mentions the possibility that the query plan or the query tree could be missing. But isn't it the case that one is always constructed before the other for query evaluation?

O5) There's no discussion of how the join order for TTJ is determined. Section 5.1 simply states that each algorithm uses the same DP algorithm with an algorithm-specific cost model which seems to contradict the footnote given on page 8 that all cost models estimate the sum of intermediate result sizes. I would suggest to include the cost model discussion in the paper and move Section 4 (on the proof of optimality of TTJ) to a technical report.

O6) There seems to be a typo error on page 10: it's stated that a larger b_{ij} implies a quicker removal of a dangling tuple from the guilty relation. Shouldn't it be a smaller b_{ij} instead?

8. Please rank the three most critical strong points and/or opportunities for improvement that led to your overall evaluation rating, e.g., "{S1, O7, O4}"
S1, O1, O2, O3

9. Inclusive writing: does the paper follow the SIGMOD inclusive writing guidelines? See https://dbdni.github.io/pages/inclusivewriting.html for more information. If not, please provide suggestions on how to improve the writing of the paper.
I agree

10. D&I compliance
OK

11. Availability: We have asked authors of all papers to make their code, data, scripts, and notebooks available, if this is possible. You can find this information in the summary of the paper on CMT when clicking on the Paper ID next to the title. In the text box provided below, please provide (a) constructive feedback to the authors about these provided artifacts (if any), (b) how this was useful in your evaluation of the submission, and (c) what changes are needed in order to make the code/data/scripts/notebooks shareable with the community (if any).
https://anonymous.4open.science/r/sigmod25-4049/

12. Minor remarks. Use this field to describe minor remarks that are not listed as opportunities for improvement, e.g., to highlight typos, formatting problems, or minor technical issues.
(1) I find Figure 7 to be not readable due to being overly cluttered.

(2) In Table 2, the storage units should be stated explicitly.

(3) Typo errors: (a) "buhsy plan" (b) page 8: evaluaiton (c) "dominate time"

13. List required changes for a revision, if appropriate, by identifying subset of previously specified opportunities for improvement (e.g., O1, O3, O6).
O1 - O6

Reviewer #4

Questions

1. Overall evaluation
Reject

3. Novelty
Novel implementation and evaluation of previous solutions to existing problem

4. Importance: select all that apply:
SIGMOD attendees will learn something interesting from the paper

5. Summary of contribution (in a few sentences)
This paper studies how to semi-joins to filter dangling tuples of a multi-way acyclic join query, i.e., those that won't participate in the final join results. The classic Yannakakis algorithm that was proposed for acyclic join, performs two phases of semi-joins along the join tree and then one phase of join over the reduced relations. This paper proposes a way of merging the semi-join phase with the join phase, while guaranteeing

the optimal time complexity.

6. Describe in detail all strong points, labeled S1, S2, S3, etc.

S1. This paper is well written with nice figures and running examples.
S2. This paper has performed extensive experiments to study the empirical performance of TreeTracker Join and other baselines.

7. Describe in detail all opportunities for improvement, labeled O1, O2, O3, etc.

O1. My biggest concern is the novelty and technical contribution. A variant of Yannakakis algorithm has been widely studied and used in dynamic query processing [1,2]. This simplified algorithm simply performs one-phase of bottom-up semi-joins, and then retrieves the join results participated by each tuple in the root node (using hashing), like the CSP search procedure in TreeTracker Join. The idea is that after this bottom-up phase, all remaining tuples in the root node of the join tree are non-dangling. Obviously, this simplified version also achieved optimal time complexity as the classic Yannkakis.

[1] The Dynamic Yannakakis Algorithm: Compact and Efficient Query Processing Under Updates, SIGMOD 2017.
[2] Change Propagation Without Joins, VLDB 2023.

This variant is also not compared with TreeTracker Join in the experiment. From my perspective, the search phase of TreeTracker Join is essentially the combination of the bottom-up semi-join phase (when join fails) and the top-down retrieval phase (when join succeeds). It is more simple since it does not need to apply backjumping repeatedly.

Also, as shown in Figure 8, if only performing one phase of semi-join, much less tuples will be removed and this simplified version will be more attractive.

O2. This work is only limited to acyclic join queries, which is very limited in practice. It is unknown how these tricks can be applied to a more comprehensive set of relational operators, e.g. outer joins, projection, group-by, and aggregation.

O3. The optimality analysis and guarantee is not surprising and can be derived from the classic theory of acyclic joins.

8. Please rank the three most critical strong points and/or opportunities for improvement that led to your overall evaluation rating, e.g., "{S1, O7, O4}"

O1, O2, O3

9. Inclusive writing: does the paper follow the SIGMOD inclusive writing guidelines? See https://dbdni.github.io/pages/inclusivewriting.html for more

information. If not, please provide suggestions on how to improve the writing of the paper.

I agree

11. Availability: We have asked authors of all papers to make their code, data, scripts, and notebooks available, if this is possible. You can find this information in the summary of the paper on CMT when clicking on the Paper ID next to the title. In the text box provided below, please provide (a) constructive feedback to the authors about these provided artifacts (if any), (b) how this was useful in your evaluation of the submission, and (c) what changes are needed in order to make the code/data/scripts/notebooks shareable with the community (if any).

yes

12. Minor remarks. Use this field to describe minor remarks that are not listed as opportunities for improvement, e.g., to highlight typos, formatting problems, or minor technical issues.

Figure 4 and 7 are very hard to read with too small fonts, legends, and patterns.