SUBMISSION: 6
TITLE: Treetracker Join: A Composable Physical Operator that Computes Join and Semijoin at the Same Time

---------- REVIEW 1 ----------
SUBMISSION: 6
TITLE: Treetracker Join: A Composable Physical Operator that Computes Join and Semijoin at the Same Time
AUTHORS: Zeyuan Hu and Daniel Miranker

---------- Overall evaluation ----------
SCORE: -1 (weak reject)
----- TEXT:
The authors present a new join algorithm for acyclic conjunctive queries based on backtracking. They prove the algorithm to be correct and instance-optimal (O(|input| + |output|), and demonstrate its efficiency with experiments.

Summary

Strengths:
1. The algorithm is very simple.
2. The experimental evaluation is comprehensive.

Weaknesses:
1. The presentation of the ideas and the proofs obfuscate the simple essence.
2. Lemma 8, which is rather important, is wrong, but salvageable.
3. It is debatable whether or not the experiment really show the practical efficiency of the algorithm.

Detailed comments

Lemma 8. For any left-deep plan without cross-product for acyclic queries, there exists a
 T_Q satisfies the join tree assumption (Definition 6).

This is not true. Consider the query Q(x, y, z) :- R(x, y), S(y, z), T(z, x), U(x, y, z), and the left-deep plan [R, S, T, U]. According to Definition 6, R is the root, and the parent of S must be R. The parent of T can be either R or S, and the parent of U is one of R, S, and T. None of the possible trees is a valid join tree.

Instead, we should go from a join tree to a left-deep plan. I claim we can convert any join tree into a left-deep plan, by reversing the GYO reduction that produced

the join tree. Of course we can no longer satisfy the property in Definition 6, but the plan will suffice for the correctness & optimality of the algorithm, because all the join attributes at each inner relation are contained in its parent in the join tree.

This however creates a wrinkle in the experimental evaluation: if there is a good plan for hash join, it may no longer have a valid join tree for TTJ. You'd have to use a different left-deep plan, and this is no longer apple-to-apple. This also shows the SSB queries are too simple: all of them have left-deep plans that correspond to a very shallow tree, so they didn't trip over the issue pointed out here.

1. Introduction

You mentioned later that if there are already no dangling tuples, TreeTracker join is identical to Hash Join and does no more work. This is an important detail worth mentioning very early on.

57: mention WCOJ is not optimal for acyclic queries.

57-59:
> One goal, similar to the efforts inspired by YA, is to eliminate the introduction of additional operators and their concomitant overhead. Much of this research has met with success
Please provide citations.

Example 1:
I find this example very confusing. First, the code is not a valid Datalog program, because the first 2 rules are not monotone and will not terminate. Second, it's not clear what's already computed and what still needs to be computed. For example Join_0 is an input to rule 3 and 4, but rule 4 also computes DTT_0.

Figure 1:
At this point it's not clear how the join tree relates to the join plan. If you don't want to explain that here, at least reference the section that defines the relationship.

90:
> The negation in Rule (3)
There is no negation in that rule.

94-95:
> one may anticipate that on each cycle
It's not clear how you run these rules in cycles.

2. Running example
114-124: This paragraph is too informal to clarify anything, but also too long to just explain the informal concepts. I suggest cutting it, and defer the formal definitions to later (or omit them if not necessary). Instead, you can avoid introducing terminology when explaining the running example, and just focus on the concrete execution of the algorithm.

122: The anti-semijoin symbol has not been introduced.

166, 167, 171: again, you used the terms "detection relation", "guilty relation", "no-good list" which do not help the reader understand the running example. Defer them until after you define them.

3. Preliminaries

260: Should "i in [k]" be "i in [k-1]"?

4. TreeTracker Join Operators

Algorithm 4.1 line 8: what's the difference between MatchingTuples being nil and being the empty set?

Experiments:

It's very impressive that you basically implemented a full-blown database for the experimental evaluation, and also implemented each baseline algorithm yourself. However, implementing the baseline yourself always runs the risk where the baseline algorithms do not receive equal attention and optimization as your new algorithm. I'd like to see at least one baseline using a state-of-the-art DB implementation, just for calibration. I recommend DuckDB for this purpose.

You highlight SSB in the main body of the paper, which is a strange choice. Star schema is too simple and your algorithm essentially just becomes probing the small dimension tables with the fact table. In your own words:

> Furthermore, in this setup, TTJ is indeed
> reduced to combining the bottom-up semijoin pass and join pass into one 5, which makes
> TTJ equivalent to YA+.

The following claim is also strange:

> However, TTJ outperforms YA+, which shows that TTJ is more
> empirical efficient than YA+ despite the equivalent formal runtime guarantee.

If TTJ degrades to the same algorithm as YA+, why is it faster then?

---------- REVIEW 2 ----------
SUBMISSION: 6
TITLE: Treetracker Join: A Composable Physical Operator that Computes Join and Semijoin at the Same Time
AUTHORS: Zeyuan Hu and Daniel Miranker

---------- Overall evaluation ----------
SCORE: -1 (weak reject)
----- TEXT:
The paper presents a new optimal algorithm, in data complexity, for computing acyclic conjunctive queries. It borrows ideas from the seminal Yannakakis three pass semijoin algorithm for acyclic conjunctive queries in that it also removes dangling tuples. This removal does not happen in seperate phases but at the moment a tuple is detected to dangling. In essence the proposed algorithm, by way of the TTJ operator, executes as a hash join over a left-deep query plan in a pipelined fashion where individual tuples that do not contribute to the overall join (as in a specific loop no matching tuple can be found) are removed - these are dangling. Such tuples are removed by removing them from the hash table. For the initial relation, no hashtable is built, and an additional ng (no-good) list is maintained so that the can be skipped during the scan. The authors prove the correctness of their algorithm and to obtain the optimal complexity (linear in input + output) require the notion o
f clean state. The reason being that the new algorithm does not remove all dangling tuples (no full reducer) and needs "clean state" to argue that the optimal complexity is reached. The paper further shows some empirical evidence that TTJ can improve over existing join algorithms.

While I think I understand the main gist, the in general paper is hard to follow. The introduction is rather confusing. For me, Example 1 did not help my understanding of TTJ. Actually, example 1 simply does not make sense to me. It seems to be introduced to argue that TTJ can be integrated with conventional joint algorithms. The running example in Section 2, on the other hand, is very instructive. But there is no relationship between the two examples and the notation used in Example 1 is not used in the remainder of the paper. The technical development is at times a bit ad hoc and could benefit from more streamlinig. The introduction of notation is spread out through the paper.

Some detailed comments:
- l44: otherwords
- l46,47: why is the notation R' and R* introduced here? It is not used in the intro.
- l106: call we call
- l285: space after Algorithm 4.1

---------- REVIEW 3 ----------
SUBMISSION: 6
TITLE: Treetracker Join: A Composable Physical Operator that Computes Join and Semijoin at the Same Time
AUTHORS: Zeyuan Hu and Daniel Miranker

---------- Overall evaluation ----------
SCORE: -2 (reject)
----- TEXT:
The paper presents a new algorithm, denoted by TTJ, for evaluating acyclic joins, i.e., joins having a join tree.

To compute such a join, the well-known Yannakakis' algorithm first applies a full reducer to remove dangling tuples. Such a full reducer executes in two phases: in the bottom-up phase, each parent relation is restricted to its tuples that join with all children relations; in the subsequent top-down phase, every child relation is restricted to its tuples that join with its parent relation.

An initial observation made in the current paper is that the top-down phase can be omitted if the join is computed top-down after the bottom-up phase. In this way, every intermediate join result will still be a projection of the final join, and hence no intermediate join will be larger than the final result.

The current paper introduces an algorithm that differs from conventional approaches by integrating the computation of dangling tuples with the join computation, eliminating the need for a separate pre-computation phase dedicated to their removal.

The experiments show that the new algorithm outperforms existing algorithms.

Unfortunately, the readability of the explanations and the quality of the technical treatment are substandard, and clearly not up to the professional standards required for ICDT. Despite serious efforts, I was unable to understand and

appreciate the details of the contribution. The new algorithm is not exposed in a declarative way, but primarily defined by a procedural code (pages 8 and 9); the illustrative examples lack clarity. For instance, I was already unable to understand Example 1. That example applies datalog syntax for explaining the algorithm in a non-procedural fashion, which seems a good idea; however, the explanation in terms of cycles is unclear.

S1: Topic of great interest to database theory.

S2: The experimental results indicate that the proposed algorithm outperforms existing algorithms.

W1: The quality of the presentation and theoretical development is substandard.

DETAILED COMMENTS AND QUESTIONS

Yannakakis' algorithm also applies to computing the projection of an acyclic join. Can TTJ be extended to deal with projections?

I found that several sentences are quite complex and difficult to understand. Take, for example, lines 50-53, stating "The YA is foundational for a preponderance of research […] may not be included."

The paper contains typos. For example, line 90, "The negation in rule (3)". There is no negation in rule (3).

Line 182. Why are you assuming bag semantics? The performance guarantees provided by Yannakakis' algorithm hold true for the standard set semantics.