##########
Reviewer 1
##########

-1: (weak reject)
This paper considers the problem of designing acyclic join algorithms that are "compatible" with existing RDBMSes by positing the following two constraints that come from practice for such join algorithms:

1. All operations should be mapped to basic operations in existing RDBMSes. In particular, only two-way join operators are allowed.
2. The algorithm should avoid a multi-phase algorithm structure.

There has been a fair bit of work on Worst-case optimal join algorithms recently but while those algorithms have optimal runtime guarantees for _any_ join query, they do not satisfy property 1 above since they work directly with multi-way joins-- it is known that for cyclic queries, one can only have _sub-optimal_ algorithms that satisfy property 1. However, the classical result of Yannakakis on input-output optimal join algorithm for acyclic queries can easily be implemented by algorithms with property 1. However, the algorithm inherently needs a 3-phase implementation (the first two are "bottom-up" and "top-down" semi-join passes and the last pass generates the output). Even more recent acyclic join algorithms in the I/O model do not satisfy property 2.

This paper presents an algorithm dubbed TrackerTree Join (or TTJ) is an algorithm that has the same runtime complexity of Yannaksis' algorithm on acyclic join queries but _does_ satisfy _both_ properties 1 and 2 above.

The result of the paper (at least at a high level) follows from the following two step observation:

(i) If the query tree plan used by Yannakasis' algorithm is "left deep" then it is not hard to see that the one does not need two semi-join passes: indeed just doing a top-down semi-join pass is sufficient. (Note that since the query tree is binary, property 1 is satisfied but property 2 is not since this is still a two-phase algorithm).

(ii) The paper then observes that even the two-phases of the algorithm can be merged into one (and hence property 2 can be satisfied as well).

The first observation is fairly easy (and I _think_ has been made in the literature before) and the bulk of the technical work in the paper is in step (ii).

I have not read the appendix of the paper but most of the details are in the first 8 pages. I have read the first 8 pages reasonably carefully and the arguments look correct to me. The paper is reasonably well-written (though I found some parts of the algorithm pseudocodes hard to follow-- see the detailed comments for more on this).

The paper does a good job of clearly stating two properties that are important in practice and then presents an algorithm that achieves the stated goal. However, the results in the paper seems to need to restrict the query plan to be "left-deep" only. First of all, I think this restriction should be stated upfront in the introduction itself. But this should be an easy presentation fix. However, I find this assumption to be a (big) weakness of the paper:

(*) The reason the Yannakakis algorithms needs two phases of semi-joins is precisely because it works for _any_ query tree (and not just a left-deep query plan). In fact, I do not see how the results in this paper can be generalized to handle arbitrary query tree (of course still for acyclic join queries). Perhaps this is possible but if so, I'd like to see details of that argument to be in the paper. And indeed with this generalization my opinion of the paper would change completely.

(*) The reason why I find this restriction to be a bit disappointing is that the paper builds up this need to design algorithms that can exploit (and not change) the decades of optimizations that are present in existing RDBMSes but it seems like restricting one to only left-deep query plans means that the proposed algorithm is not able to exploit various query optimizations that might need the full generality of query plan trees.

(*) To be fair, the paper does mention that left-deep query plans are used in practice but to me it seems like this restriction seems like could be an important one. In particular, if this is not the case, then the paper needs to do a _much better_ job of advocating _why_ this restriction is not a big deal.

Given the above weakness, I would recommend a weak reject for the paper.

Detailed Comments for the Author(s)

Some of the comments below are minor presentation suggestions but others might need more work:

[Lines 115-116] I think property (2) is not that well defined. It would be useful to perhaps formalize what you mean by "multi-phase algorithm structure"-- otherwise this could potentially become a matter of "taste." Similarly property (1) could be formalized a bit more (but I think the way property (1) as stated currently is specific enough).

[Algo 1.1 + General Comment] Inputs are passed by reference in your algorithms and in some of the algorithms the modification of these inputs is crucial. I think it would be easier to follow the algorithms if in addition to the input and output, the pseudocode also mentions _how_ the inputs are changed (at least for those parameters for which the change matters).

[Lines 291, 295] The periods after the \Join are a bit awkward. In my first read I thought "\Join_k." and "\Join_u." were some new operators.

[Line 339] The operator \overline{\Semi-Join} is not defined.

[Sec 4] An English description of the what Algos 4.1-4.3 do will be useful I think.

[General comment] $\ell$ typesets much better than $l$

[Algo 4.1] Is there a return statement missing?

[Sec 5 algos] I liked that in Sec 4 the global variables were explicitly mentioned-- it would be good to do the same in this section as well.

[Line 680] "Denote" -> "denote"

##########
Reviewer 2
##########

-2: (reject)
# Summary:

The paper proposes a novel join algorithm for acyclic join queries (join trees). The algorithm is optimal in the sense of Yannakakis' algorithm. However, the authors propose to avoid the full reduction in the first pass by interleaving/combining the identification of dangling tuples (for semi-join reductions) and the actual join computation.

# Strengths:

S1: As mentioned by the authors, join algorithms are central to relational databases, and improving upon them remains an important problem. The study of optimal join algorithms is important.

S2: Overall there are some interesting ideas here that may translate into algorithms that are more efficient in practice for acyclic joins.

# Weaknesses:

W1: The paper appears to be off-topic for PODS. The contribution here is not theoretical, in that the proposed join algorithm has the same data complexity as the Yannakakis' algorithm (and is not more elegant). Rather the authors argue that the benefits of the proposed approach are practical: that it would be more compatible with existing relational database infrastructure, and potentially more efficient. The contributions that the authors claim thus do not appear to be theoretical, but rather practical.

W2: The claims of practical benefits are not evaluated in any way. No experimental evidence is provided to show that the proposed join algorithm is more efficient.

W3: The motivation for the new join algorithm is difficult to follow. The novelty is unclear. Some claims made regarding existing work, and the improvement over existing work, are not justified, or appear to be imprecise. As examples:
- "Yet these formal algorithmic improvements rarely make their way into fielded general purpose relational query systems." I'm not sure I understand the point. Worst case optimal join algorithms are relatively new, and are gaining traction, and form part of the offering of companies such as RelationalAI.
- "Query systems must then represent, and optimize query plans containing traditional unary and binary operators then add k-ary operators." It is not clear to me that this is a problem (or why it is a problem). Joins are commutative and associative, so one could argue that n-ary join operators are a more natural representation of what the query expresses.
- "At least one effort determined it was best to completely abandon the relational algebra approach and start from scratch [1]." It is not clear what it means to "abandon the relational algebra approach". EmptyHeaded [1] applies logical query plans, Yannakakis' algorithm, etc. So at least to me it seems to keep with the tradition of the relational algebra approach.
- "We suggest two algorithmic elements that are practical constraints" Why are these constraints considered practical? I do not find evidence in the paper that they lead to more "practical" implementations of database engines.
- "Empirically and without proof of correctness of the system, EmptyHeaded [1] (Section 3.5) eliminates the top-down pass of Yannakakis’s algorithm and demonstrates a 10% performance improvement for tested workload." This is only in certain cases; they eliminate the top-down pass "if all the attributes appearing in the result also appear in the root node" [1].

W4: The text and notation appear imprecise. Notation is used before it

is introduced. Different types of notation are used interchangeably, and the notation is often difficult to follow. This, in turn, makes it difficult to follow the proposal in detail. As some examples of imprecision:
- "In this paper, the big-O notation is in data complexity ignoring terms that depending on query expression not data, and big-O indicates the combined complexity" Is this not a direct contradiction?
- "O(n + r)" What is n and what is r? (These are introduced later but need to be introduced when first used.)
- "such that if a duplicate tuple of s shows up again" Is S a base relation? This only happens if base relations have bags of tuples?
- "to denote reducing semi-join program on G_Q" What is G_Q? (This is introduced later but needs to be introduced when first used.)
- "the root of G_Q is the left-most relation at the bottom." I don't understand. Would it not be the top node if left-deep? The left-most relation at the bottom would be a leaf?
- "Algorithm 1.1 is enumerating output top-down over G_Q" But Algorithm 1.1 just describes a binary join? I don't see how it is enumerating top-down?
- "apply P_{Q,1} on G_Q" What is P_{Q,1}?
- "and root R_1". Before the root of G_Q was defined as ⋈_k.
- "perform pairwise join from root R1 to leaves recursively" Which relation is S, and which is R?
- Corollary 5.8 mentions that the proposed join algorithm and Yannakakis' algorithm are equivalent "from both scope of applicability and algorithmic complexity". What does "scope of applicability" mean? How could it be proved?

# Recommendation:

Overall, I think that the potential contributions of this paper are stated in terms of practical benefits, rather than theoretical understanding. Hence I think the paper is off-topic for PODS, and recommend a reject. I think that the paper would be better targeted to a systems-oriented database conference, but first the writing and notation would need to be sharpened up; novelty, contributions and claims would also need to be stated more precisely; and experiments would need to be added to validate the (sometimes implicit claims) of practical benefits..

# Minor comments:

- "we show [that] a composition"
- You can use $\ltimes$ for semi-join, rather than drawing it.
- "that [are] below"
- Figure 1 is too difficult to read.

##########
Reviewer 3
##########

-1: (weak reject)
The paper claims to propose an algorithm for evaluating acyclic conjunctive queries that
avoids the semijoin phases, and k-way joins. The motivation is "better integration with existing DBMS".

The goal of the paper is unlcear to me. The abstract and opening sentences talk about acyclic joins.
Then, they talk about WCOJAs whose goal is to evaluate queries in time polynomial in the size of the input + output.
But the Yannakakis algorithm for acyclic joins is already worst-case-optimal (the authors are aware of this as they cite [42]).
Then, they refer to some struggles with the integration of the Yannakakis alg., referring to [47] but do not explain how, exactly, semijoins affect finding good query plans for acyclic CQs. Finally, the Yannakakis algorithm involves only binary joins,
so item (1) of the "practical constraints" is also left unmotivated.