Efficient Ranked Access over Joins

Nikolaos Tziavelis

Supervised by Prof. Mirek Riedewald and Prof. Wolfgang Gatterbauer
Northeastern University, Boston, MA, USA

Abstract

Join queries over multiple tables with many-to-many relationships (e.g., in graph analytics) can produce a huge output that is infeasible to compute. However, users may have particular preferences over the answers in the output, and may be interested in accessing only a small subset according to that ranking; either to retrieve the most important answers or the quantiles for a statistics summary. The thesis of this proposal is that for many such queries, access patterns, and ranking functions over the answers, ranked access can be performed efficiently, without first computing the output of the join. This is captured theoretically by non-trivial complexity guarantees and shown in practice with significant performance improvements over typical implementations in state-of-the-art database management systems (DBMSs) that sort the join output. We have so far given algorithms and complexity guarantees for the problems of ranked enumeration, direct access, and quantiles over equi-joins, as well as joins with complex inequality predicates. To complete the picture, we plan to extend our results to support more expressive queries, distributed computation, and stronger complexity guarantees toward instance-optimality. Besides addressing fundamental questions regarding the limits of query processing, this work opens up unexplored possibilities for the design of DBMSs. We clearly demonstrate how existing systems fall short in handling ranking over joins efficiently, with our algorithms outperforming them by orders of magnitude.

1. Problem Focus

Join Queries. Joins are fundamental in data-processing systems, as they enable the composition of data from multiple sources. They are also notoriously critical for performance, as they can produce *huge intermediate or final results*. This is especially true when dealing with graph data [1], but can also occur with traditional relational data if many-to-many relationships exist across tables or non-equi-join conditions are used. A breakthrough result in our understanding of join processing was that the worst-case complexity of traditional (binary) join implementations is suboptimal [2], which led to a number of Worst-Case Optimal Join (WCOJ) algorithms that fill this gap [3, 4]. Besides, work on enumeration [5, 6] has focused on returning a stream of answers as fast as possible even if the full output is too large to compute. These efforts toward strong worst-case guarantees have not only created excitement about their theoretical value, but are also aligned with the pervasive goal of database systems to ensure predictable performance for complex queries or skewed data.

Ranking Join Answers. Users may *prefer some join answers over others* based on their importance or relevance. For instance, higher importance may be assigned to newer (time) or more reliable (quality) data. We formalize this by a ranking function that establishes a total order over the answers. A common example is SUM where database tuples are assigned weights and the weight of a query answer is the sum of weights of the contributing tuples. Sorting the answers is clearly expensive; even if we were to compute the join efficiently, all join answers must be produced to be sorted. For a database of n tuples and ℓ relations, the join output size is $\mathcal{O}(n^\ell)$.[1] The question we ask is whether we can retrieve a few select answers (according to the ranking) *without materializing and sorting the entire output*, or in other words, whether we can "push" the ranking deeper into joins. This turns out to be not as simple as reordering the two operations. For example, the top-10 tuples from each joining relation do not necessarily produce the top-10 join answers, and may not even join at all. Achieving our goal requires *novel algorithms* where joining and ranking are interleaved.

Example 1. *To study bird interactions, an ornithologist uses a bird observation dataset B(Species, Family, Cnt, Latitude, Longitude) and wants to extract pairs of observations for birds of different species that have been spotted in the same region. Pairs with the greatest combined number of birds (Cnt) should also appear first:*

```
SELECT   *, B1.Cnt + B2.Cnt as Weight
FROM     B B1, B B2
WHERE    B1.Species <> B2.Species
  AND    ABS(B1.Latitude - B2.Latitude) < 1
  AND    ABS(B1.Longitude - B2.Longitude) < 1
ORDER BY Weight DESC
```

This query contains inequality ($<$) and non-equality ($<>$) predicates and its answers are ranked by SUM. The output size is $\Omega(n^2)$ in the worst case, thus, regardless of the join implementation, sorting will take time $\Omega(n^2 \log n)$. Yet, our new algorithms can retrieve the top-10 answers or even the median answer asymptotically faster, in time $\mathcal{O}(n \operatorname{polylog} n)$.

VLDB 2023 PhD Workshop, co-located with the 49th International Conference on Very Large Data Bases (VLDB 2023), August 28, 2023, Vancouver, Canada
tziavelis.n@northeastern.edu (N. Tziavelis)
0000-0001-8342-2177 (N. Tziavelis)

[1] Tighter bounds exist if we take the structure of the query into account [2], but we prefer to keep things simple here.

1.1. Ranked Access Patterns

We consider a few different ways that a user may want to access the ranked result of a join query.

Top-k. The *top-k* paradigm asks for the the first k ranked answers for a given k, which is typically considered to be a relatively small constant. Since the value of k is known, it can be exploited for pruning.

Ranked Enumeration. *Ranked enumeration* asks for the answers to be returned in order, one-by-one until all of them are returned or the user stops the procedure. It generalizes top-k because the value of k is flexible and not fixed in advance. For this reason, we introduced the term "any-k" for this paradigm. This functionality can be useful for exploratory analysis tasks, where setting k without first seeing some answers is difficult. The goal is to provide complexity guarantees for *every possible value of k*. We denote by $\mathrm{TT}(k)$ the time required up to the k^{th} answer, which should be as small as possible no matter if k is small or equal to the join output size.

Direct Access. A more general access pattern is *direct access*, where the indexes requested from the sorted array of answers do not need to be consecutive integers that start from 0 as in ranked enumeration, but can be arbitrary (e.g., all q-quantiles). The goal is to build a data structure that can support all these accesses efficiently.

Quantiles. Quantile queries (or a variant called "selection"[2] ask for only one access, such as the median answer. Compared to direct access, we are not required to handle multiple accesses and no data structure for future accesses needs to be constructed.

1.2. Contrast to Prior Top-k Joins

A well-known algorithm for top-k joins is the instance-optimal Threshold Algorithm by Fagin et al. [7]. However, it only works for a restricted class of 1-to-1 joins. Follow-up work extended the algorithm to more general joins [8, 9] but under a "middleware" cost model where cost is measured in terms of distinct tuples accessed, while *join cost is not taken into account* [10]. This is in contrast to the line of work on WCOJ or enumeration, which focuses on the RAM model of computation where the primary concern is to avoid unnecessary joins and large intermediate results. Our work aims to bridge this gap by adopting the RAM model.

2. Highlights of Current Results

2.1. Any-k Algorithms for Equi-joins

For acyclic equi-joins, where join predicates are restricted to equalities and relations can be organized in a join tree, we designed any-k algorithms [11] for SUM with strong guarantees: If the query size is constant, we were able to achieve $\mathrm{TT}(k) = \mathcal{O}(n + k \log k)$ for an input database of n tuples. This is optimal since it takes $\Omega(n)$ to read the database and $\Omega(k \log k)$ to return k answers sorted. Remarkably, the user starts seeing the top answers after only linear time, even if the join output is much larger. For the case where query size is non-constant, we proposed an algorithm [12] that achieves the best-known complexity. [3] Surprisingly, it is asymptotically faster than (generic, comparison-based) sorting for returning the entire output. This is possible because the query answers are not independent, but have a shared structure that the algorithm exploits. Our algorithm does not only apply to joins, but also to ranked enumeration of source-target paths in a DAG, and even more generally, to any problem solvable by Dynamic Programming. Beyond SUM, other ranking functions are supported if they satisfy appropriate monotonicity properties [12].

Figure 1a illustrates the advantage of any-k over the approach of current DBMSs which follow the "JoinFirst" approach of applying ranking after the join. The query in this experiment is a join of 4 relations in a chain over uniform synthetic data. Each relation contains 10^4 tuples and each join value appears 10 times on expectation.

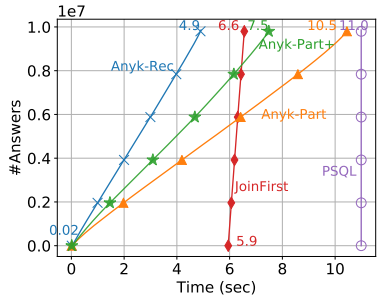
2.2. Inequality Predicates

While our initial work focused on equality conditions, we then extended it to more general join conditions [14]. For acyclic joins with any conjunctions or disjunctions of inequality predicates between adjacent relations in the join tree, any-k is possible with $\mathrm{TT}(k) = \mathcal{O}(n \operatorname{polylog} n + k \log k)$, i.e., within a polylogarithmic factor of the equi-join guarantee. The key insight is a "factorized" representation of the output which essentially reduces the inequality-join to an equi-join over (polylogarithmically) larger relations.

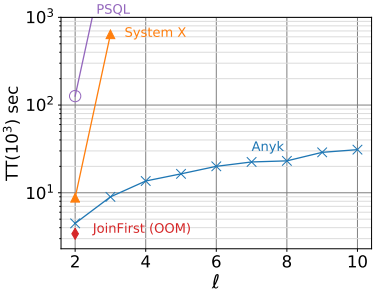
Our result is quite surprising, given that existing DBMSs struggle to handle complex join predicates like those in Example 1 even without the ranking. Our algorithms handle both simultaneously and outperform DBMSs by orders of magnitude. Figure 1b illustrates this in terms of the time to find the top-1000 answers for a path query on the RedditTitles [15] temporal graph. The $572k$ edges in the dataset represent posts from a source community to a target community identified by a hyperlink in the post title. The query returns paths of length ℓ with increasing timestamps, decreasing sentiment (i.e., a sign of negative emotion propagation), and ordered by the SUM of readability scores.

[2] The two problems differ in whether a percentage or an absolute index is given as the input.

[3] This algorithm (anyK-part+) asymptotically dominates other algorithms like anyK-rec, which was also proposed by Deep and Koutris [13] concurrently with our initial work [11].



(a) The "JoinFirst" approach of sorting all answers needs 5.9 sec for the top answer and 6.6 sec for the last. In contrast, our any-k takes 0.02 sec for the top answer (hence more than 200 times faster) and 4.9 sec for the last with a variant called "Anyk-Rec". So we get not only the first answer faster, but all answers faster. All existing DBMSs we tried, such as PostgreSQL (PSQL) here, follow an approach similar to "JoinFirst" and are outperformed.



(b) For a path query with 2 inequality predicates on the RedditTitles dataset, our approach performs significantly better than PostgreSQL (PSQL) and a commercial system optimized for in-memory computation (System X). Our own in-memory "JoinFirst" approach runs out of memory (OOM) when the path length ℓ is more than 2.

Figure 1: Any-k experimental results.

2.3. Dichotomy Theorems

For both direct access and quantile queries under SUM, we precisely characterized the (self-join-free[4]) join queries that can be handled efficiently, and gave algorithms for those cases [17]. We consider $\mathcal{O}(n \operatorname{polylog} n)$ time as efficient, since it is close to the database size n and independent of the join output size. We showed that, under certain hardness assumptions in fine-grained complexity, constructing a data structure for direct access is possible only for trivial cases. By relaxing the task to that of quantiles where only one access is required, then any binary join can be handled efficiently. For all other acyclic queries, which are provably hard, we showed that efficient (deterministic) approximation of quantiles is possible [18].

2.4. Projections

Our results also cover join queries with projections, formalized by *conjunctive queries*. For ranked enumeration, we extended [12] a dichotomy of Bagan et al. [5] that was developed for unranked enumeration [5], showing that the class of queries that can be handled efficiently (i.e., with $\mathrm{TT}(k) = \mathcal{O}(n + k)$ ignoring logarithmic factors) is precisely the same; it is those queries that are "free-connex", a certain restriction on the allowed projections. Thus, the additional requirement of an ordered output does not change the tractability landscape and only costs a logarithmic factor. We note that follow-up work by Deep et al. [19] has considered more expensive time guarantees for queries beyond the free-connex class, in terms of preprocessing and delay between answers. [6] Conjunctive queries are also considered in our study of direct access and selection [17].

3. Future Directions

3.1. Supported Queries

We have demonstrated how existing systems are lacking in terms of support for join queries with ranking, by focusing mainly on conjunctive queries. A major next step is to extend our study to more expressive queries, such as those with recursion [21] or other types of operators.

For the supported ranking functions, while we have algorithms for some of the most common cases, we are lacking in terms of hardness results for others. Can the functions that are outside of our tractable classes (e.g., using MEDIAN as a weight aggregation function) be proven to be intractable? Or are there still cases potentially useful in practice that remain unexplored?

3.2. Distributed Computation

Our algorithms assume an in-memory computation model. To make them more useful for big-data processing, we aim to explore techniques to distribute computation to many machines. Popular practical frameworks for this task include Hadoop MapReduce or Spark which partition computationally intensive tasks into smaller, independent tasks to be executed in parallel. On the theoretical side, the MPC model has been used to analyze distributed joins [22]. An intriguing question is whether the the distributed computation setting requires new algorithms and techniques that are fundamentally different than the sequential case.

[4] This restriction is common in hardness results in this area [16].

[5] Again, this is under certain hardness hypotheses and for queries without self-joins.

[6] We have argued that preprocessing and delay are less practically relevant than $\mathrm{TT}(k)$ as measures of success for enumeration. [20]

3.3. Toward Instance Optimality

The guarantees we have explored so far are concerned with worst-case time and space complexity, and it is open whether stronger guarantees are possible. As aforementioned, the Threshold Algorithm [7] is instance-optimal in the middleware model, a very strong guarantee which partly led to the 2014 Gödel Prize for Fagin, Lotem, and Naor. In addition to the simpler join model, instance-optimality crucially relies on the assumption that input relations are given sorted. This assumption makes it possible to retrieve the top-k tuples without even reading the entire input. In contrast, our algorithms (and other WCOJ) unavoidably have $\Omega(n)$ as a lower bound since the winning tuples could be anywhere in the input. Thus, an open question is whether stronger guarantees are possible under this assumption in the RAM model.

References

[1] S. Sahu, A. Mhedhbi, S. Salihoglu, J. Lin, M. T. Özsu, The ubiquity of large graphs and surprising challenges of graph processing: extended survey, VLDB J. 29 (2020) 595–618. doi:10.1007/s00778-019-00548-x.

[2] A. Atserias, M. Grohe, D. Marx, Size bounds and query plans for relational joins, SIAM Journal on Computing 42 (2013) 1737–1767. doi:10.1137/110859440.

[3] H. Q. Ngo, E. Porat, C. Ré, A. Rudra, Worst-case optimal join algorithms, J. ACM 65 (2018) 16. doi:https://doi.org/10.1145/3180143.

[4] T. L. Veldhuizen, Triejoin: A simple, worst-case optimal join algorithm, in: ICDT, 2014, pp. 96–106. doi:10.5441/002/icdt.2014.13.

[5] G. Bagan, A. Durand, E. Grandjean, On acyclic conjunctive queries and constant delay enumeration, in: International Workshop on Computer Science Logic (CSL), 2007, pp. 208–222. doi:10.1007/978-3-540-74915-8_18.

[6] N. Carmeli, S. Zeevi, C. Berkholz, A. Conte, B. Kimelfeld, N. Schweikardt, Answering (unions of) conjunctive queries using random access and random-order enumeration, TODS 47 (2022). doi:10.1145/3531055.

[7] R. Fagin, A. Lotem, M. Naor, Optimal aggregation algorithms for middleware, JCSS 66 (2003) 614–656. doi:10.1016/S0022-0000(03)00026-6.

[8] I. F. Ilyas, W. G. Aref, A. K. Elmagarmid, Supporting top-k join queries in relational databases, VLDB J. 13 (2004) 207–221. doi:10.1007/s00778-004-0128-2.

[9] A. Natsev, Y.-C. Chang, J. R. Smith, C.-S. Li, J. S. Vitter, Supporting incremental join queries on ranked inputs, in: VLDB, 2001, pp. 281–290. URL: http://www.vldb.org/conf/2001/P281.pdf.

[10] N. Tziavelis, W. Gatterbauer, M. Riedewald, Optimal join algorithms meet top-k, in: SIGMOD, 2020, pp. 2659–2665. doi:10.1145/3318464.3383132.

[11] N. Tziavelis, D. Ajwani, W. Gatterbauer, M. Riedewald, X. Yang, Optimal algorithms for ranked enumeration of answers to full conjunctive queries, PVLDB 13 (2020) 1582–1597. doi:10.14778/3397230.3397250.

[12] N. Tziavelis, W. Gatterbauer, M. Riedewald, Any-k algorithms for enumerating ranked answers to conjunctive queries, CoRR abs/2205.05649 (2022). doi:10.48550/ARXIV.2205.05649.

[13] S. Deep, P. Koutris, Ranked enumeration of conjunctive query results, in: ICDT, volume 186, 2021, pp. 5:1–5:19. doi:10.4230/LIPIcs.ICDT.2021.5.

[14] N. Tziavelis, W. Gatterbauer, M. Riedewald, Beyond equi-joins: Ranking, enumeration and factorization, PVLDB 14 (2021) 2599–2612. doi:10.14778/3476249.3476306.

[15] S. Kumar, W. L. Hamilton, J. Leskovec, D. Jurafsky, Community interaction and conflict on the web, in: WWW, 2018, pp. 933–943.

[16] P. Koutris, J. Wijsen, Consistent query answering for primary keys and conjunctive queries with negated atoms, in: PODS, 2018, p. 209–224. doi:10.1145/3196959.3196982.

[17] N. Carmeli, N. Tziavelis, W. Gatterbauer, B. Kimelfeld, M. Riedewald, Tractable orders for direct access to ranked answers of conjunctive queries, TODS 48 (2023). doi:10.1145/3578517.

[18] N. Tziavelis, N. Carmeli, W. Gatterbauer, B. Kimelfeld, M. Riedewald, Efficient computation of quantiles over joins, in: PODS, 2023, p. 303–315. doi:10.1145/3584372.3588670.

[19] S. Deep, X. Hu, P. Koutris, Enumeration Algorithms for Conjunctive Queries with Projection, in: ICDT, volume 186, 2021, pp. 14:1–14:17. doi:10.4230/LIPIcs.ICDT.2021.14.

[20] N. Tziavelis, W. Gatterbauer, M. Riedewald, Toward responsive DBMS: Optimal join algorithms, enumeration, factorization, ranking, and dynamic programming, in: ICDE tutorials, 2022. URL: https://northeastern-datalab.github.io/responsive-dbms-tutorial/. doi:10.1109/ICDE53745.2022.00299.

[21] M. Abo Khamis, H. Q. Ngo, R. Pichler, D. Suciu, Y. R. Wang, Convergence of datalog over (pre-) semirings, in: PODS, 2022, p. 105–117. doi:10.1145/3517804.3524140.

[22] P. Koutris, S. Salihoglu, D. Suciu, Algorithmic aspects of parallel query processing, in: SIGMOD, 2018, p. 1659–1664. doi:10.1145/3183713.3197388.