Toward Responsive DBMS: Optimal Join Algorithms, Enumeration, Factorization, Ranking, and Dynamic Programming

Nikolaos Tziavelis
Northeastern University
Boston, USA
0000-0001-8342-2177

Wolfgang Gatterbauer
Northeastern University
Boston, USA
0000-0002-9614-0504

Mirek Riedewald
Northeastern University
Boston, USA
0000-0002-6102-7472

***Abstract*— When processing join queries over big data, a DBMS can become unresponsive, i.e., it takes very long until any output tuples appear. Ranked enumeration addresses this problem by attempting to return the most important answers as quickly as possible, ideally in time that is linear (or quasilinear) in input size, even if the complete output is much larger. Aside from its practical usefulness, ranked enumeration is closely related to, and in a way unifies, several other problems involving joins. The common goal is the design of optimal algorithms that are guaranteed to avoid large intermediate results and thus achieve time or space complexity close to a lower bound. Arguably, avoiding query plans that produce huge intermediate results has been an overarching goal of database optimizers, which is part of the reason why optimal join algorithms, enumeration, and factorized representations have generated a lot of excitement. In this tutorial, we embark on an exploration of these topics, showing how they are intimately connected with a wide range of fundamental problems in computer science.**

***Index Terms*—ranked enumeration, join queries, ranking function, factorized representations**

I. Introduction

In many data science applications, a query may have a combinatorially large number of answers or intermediate results, especially if it involves joins of more than 2 relations. As a consequence, traditional join techniques can take too long until they return any tuple to the user. To tackle this issue, approaches like enumeration have emerged that aim to return some answer early, followed by all the others in quick succession [1], [2]. Enumeration is particularly useful in the presence of some notion of *importance*, i.e., a way to decide which answers are preferred over others. This paradigm of query answering is called *ranked enumeration* [3], and it may avoid wasting resources on low-ranked answers. This is similar in spirit to classic top-k algorithms [4], but now k does not need to be known in advance and the goal is to give strong guarantees for time and space complexity.

Solving the general problem of ranked enumeration is challenging because, in addition to joins, the query may involve other operators such as selection, projection, union, and aggregation, or complex join predicates such as inequalities. These challenges provide an opportunity to survey classic results and recent developments, such as those in the theory of optimal join algorithms [5], [6], enumeration [1], [7], and factorized representations of query results [2], [8]. We revisit several of these areas, which all aim for optimality and contain necessary ingredients to the unifying problem of ranked enumeration. To get a sense of the fascinating questions that arise when looking at these areas through the lens of ranked enumeration, consider the widely used Dynamic Programming (DP) approach [9]. The strength of DP is an efficient search for the best answer in a combinatorially large search space. The question ranked enumeration asks is: how can we efficiently find the second-best answer, then the third-best, and so on?

II. Tutorial Information

Audience. This tutorial is aimed at anybody who has ever worried about the time it takes for their query to return any answer. Long delays are quite common when joining large input, especially when more than 2 relations are involved or when the join predicate contains conditions other than equality. In addition to theoretical results with strong asymptotic guarantees, we also discuss encouraging recent empirical results [3], [8] that show the practicality of ranked enumeration and the shortcomings of existing DBMS approaches.

Prerequisites. This tutorial does not have a hands-on component and no software tools are required by the audience. To make all material accessible to those interested in the practical impact of the techniques, the tutorial will favor intuitive examples and explanations over low-level technical details. We only assume familiarity with concepts covered in typical undergraduate database and algorithms classes.

Outline. This 3-hour tutorial consists of six main parts:

1) Shortest Paths and the "Top-1" Problem
2) Acyclic Queries and Unranked Enumeration
3) Cyclic Queries
4) Ranked Enumeration
5) Factorizeed Join Representations
6) Relationship to Other Problems

2375-026X/22/$31.00 ©2022 IEEE
DOI 10.1109/ICDE53745.2022.00299

We will conclude with a variety of open research problems. Slides and videos of the tutorial will be made available on the tutorial web page.[1]

Prior Offerings. We presented a related 1.5 hour tutorial at SIGMOD 2020 [10].[2] That previous shorter version focused on the distinction between classic top-k algorithms and the recent paradigm of ranked enumeration. Aside from greater depth and breadth in covering ranked enumeration, this tutorial includes a broader range of closely related problems, including optimization problems, unranked enumeration, factorized join representations, and direct access to query query answers.

III. Tutorial Content

Throughout the tutorial, n denotes input size (e.g., the number of tuples in a database), r denotes the number of possible solutions (e.g., the number of query answers) and we generally express results in terms of data complexity.

A. Shortest Paths and the "Top-1" Problem

Dynamic Programming (DP) [9] is the archetypical approach for problems whose solutions have a shared structure. In a weighted graph, we may want to count the number of paths between 2 nodes or to find the shortest one. For DAGs, the Bellman-Ford algorithm [11] finds the shortest path in linear time using DP. The benefit of DP is that typically its running time is close to the problem's representation size (e.g., the number of edges) and that it avoids looking at all possible solutions, which can be exponentially many (e.g., the paths). The shortest-path problem is conceptually similar to *optimization* problems that ask for "the best" solution among the possible ones. We call these problems "top-1" because they have a second-best solution, and third-best and so on. These are the problems that admit ranked enumeration.

In databases, aggregation tasks over joins display a similar shared structure. For example, a query may ask for the COUNT of a join or to find the MIN over the SUM of the join answer attributes. The latter is a top-1 problem since there is a join answer with the second-best SUM. Like paths in a graph, join answers have a shared structure: In the same way that an edge may participate in several paths, a database tuple may participate in several join answers. Thus, it is not surprising that similar techniques have been developed for aggregation tasks over joins: For example, if the joining relations can be organized in a path, DP finds the top-1 SUM in time linear in database size [12], [13]. However, the path analogy does not capture the full generality of queries. Even for queries that are acyclic, the structure of joining relations can be a tree instead of a path. For these tree-structured problems, Non-Serial Dynamic Programming (NSDP) [14], a generalization of DP from paths to trees, can solve top-1 problems efficiently.

A natural question to ask is which tasks can be solved by this general algorithm. The answer lies in algebraic structures called *(commutative) semirings*, which are at the core of efficient algorithms across computer science [15]. General frameworks have been developed for incorporating any such semiring [13], [16], [17]. Going back to the MIN-SUM example, the $(\min, +)$ operators indeed belong to a semiring. The algebraic abstraction is powerful because the same algorithm can be used for many different problems by simply plugging in a different semiring. As aforementioned, ranked enumeration is restricted to top-1 problems, where the semiring needs to have an additional property called *selectivity* [18].

B. Acyclic Queries and Unranked Enumeration

A fundamental algorithm for acyclic joins is due to Yannakakis [5]: It returns the complete output in $\mathcal{O}(n + r)$ time, which is *optimal* since it is necessary to read the input and write the output. Its secret of success lies in semi-join reductions [19]: With two passes over the data, it removes all dangling tuples and guarantees that any intermediate join result can be extended to a valid output tuple. As it turns out, these semi-join reductions are also a special case of the general DP algorithm using an appropriate (Boolean) semiring.

After reducing the input relations, the Yannakakis algorithm joins them to produce the full result. A small modification to this second phase gives rise to *constant-delay enumeration* [1]. This modification requires a shift of perspective: instead of breath-first (one table at-a-time), the input tuples are traversed depth-first. This delivers some answers very quickly, while maintaining the time complexity for returning all output tuples. The situation is more tricky with projections. Bagan et al. [1] show that queries called *free-connex* allow constant-delay enumeration after linear-time preprocessing, and (under common complexity-theoretic assumptions) these are the only queries (without self-joins) that admit such an algorithm. Later work pursued similar dichotomies for more general settings such as unions of queries [20], allowing database updates [21], [22], or incorporating functional dependencies [23]. Interestingly, even though the task here is to enumerate answers in no particular order, most of these algorithms have an underlying *lexicographic order* on the attributes.

C. Cyclic Queries

Even though the optimality guarantees of Yannakakis and its adoptions to enumeration only apply to acyclic queries, the same tools can be leveraged for cyclic queries using tree decompositions [24], albeit with a complexity penalty. Originating from the concept of the treewidth of a graph [25], the key idea is to eliminate cycles by grouping multiple elements of a cycle together into a "bag" and treating them as one element. The goal is then to reduce the problem structure to an acyclic one that can be handled efficiently, only at the cost of computing these bags of elements—known as the *width* of the decomposition. In the case of queries, a bag of relations implies that their complete join output must be materialized and the width parameter determines the size of the intermediate result. A flurry of decomposition techniques with ever-decreasing width parameters have been developed, [26]–[32], with the current frontier being the *submodular-width*

[1] https://northeastern-datalab.github.io/responsive-dbms-tutorial/
[2] https://northeastern-datalab.github.io/topk-join-tutorial/

decompositions [32]. The key innovation, from a practical point of view, is that they decompose a cyclic query into a *union of multiple trees*, each receiving a subset of the input. This enables lower widths: for a query that computes 4-cycles in a graph, a single-tree decomposition must materialize $\mathcal{O}(n^2)$-size bags, while a decomposition into multiple trees is possible with $\mathcal{O}(n^{1.5})$-size bags.

Tree-decomposition techniques do not eliminate the need for cyclic-join evaluation, since each bag still needs to be materialized. Unfortunately, as Ngo et al. [6] show, for join queries with cycles the optimal $\mathcal{O}(n+r)$ bound of Yannakakis is unattainable based on well-accepted complexity-theoretic assumptions. They therefore propose the notion of *worst-case-optimal join (WCOJ)* algorithms [33] of time complexity $\mathcal{O}(n+r_{\mathrm{WC}})$, where r_{WC} denotes the size of the largest possible output of a query *over any database instance* of size n. For r_{WC}, Atserias, Grohe, and Marx [34] provide a tight upper bound by connecting join-output size to the *fractional edge cover* of the corresponding query hypergraph, now known as the AGM bound. Several WCOJ algorithms have been proposed to match the AGM bound [6], [35]–[37].

D. Ranked Enumeration

Ranked enumeration [3], [8], [38]–[43] for join queries is the problem at the center of the tutorial. A ranked-enumeration algorithm returns the join answers in the order of importance as imposed by a ranking function. Its goal is to minimize the time for returning the k top-ranked answers *for every value of* k. This paradigm generalizes the more well-known top-k and is also reminiscent of the general concept of an anytime algorithm [44]. In order to emphasize this, we refer to ranked-enumeration algorithms also as "*any-k*" algorithms as a shorthand for "*anytime top-k*" or "top-k for any k."

Due to the relationship of the top-1 problem to the shortest-path problem and DP, any-k is closely connected to ranked enumeration of paths in a weighted graph [3]. This view allows us to reveal common foundations between a variety of solutions that had been proposed in isolation, often re-inventing the wheel. We will demonstrate how most existing algorithms rely on two different major techniques. The first is the *Lawler-Murty procedure* [45], [46] that had been used in the database community to design algorithms for ranked enumeration [40] and for graph-pattern search [41], [47]. The second technique exploits a generalization of the DP principle of optimality to enumerate paths recursively [48], [49]. The same recursive call structure has been rediscovered in recent work on ranked enumeration for join queries [39]. While it was previously believed [3] that these two approaches have unique advantages and are incomparable (when query complexity is also accounted for), we will present a single unified algorithm that combines the best of both worlds [50].

We will focus on the simpler case of ranked enumeration for paths because it allows us to decouple the core algorithmic techniques of ranking from other concerns that are relevant to general queries. To lift any-k from paths to general join queries, we make full use of the toolbox discussed in other parts of the tutorial. Paths are generalized to tree structures via NSDP, cyclic queries are decomposed to trees via tree decompositions, free-connex queries are handled by eliminating the projections, and complex join predicates are *factorized* into efficient representations.

E. Factorized Join Representations

This part of the tutorial focuses on how to construct, from a database instance and given join query with complex join conditions, intermediate representations that are compact and quickly traversable. These representations are not limited to ranked enumeration, but extend to many other tasks such as answering Boolean queries, aggregates, and unranked enumeration. While earlier parts of the tutorial illustrate how these are captured by different forms of DP and shortest-paths, this section focuses on the graph construction itself. Already for equi-joins, obtaining the best possible bounds for any-k requires a careful representation that groups tuples with the same join-attribute-value together [3]. This fundamental insight, present in all equi-join algorithms such as a hash-join, is also the key idea behind factorized databases [2], [51].

This framework can accommodate more complex join predicates and gives rise to interesting questions about their most efficient representation. These predicates include inequalities [52] ($<$), non-equalities [1] ($\neq$), DNF formulas thereof [8], as well as higher-arity predicates (Not-all-equal) [53]. As an example, the inequality $A < B$ can be naively represented as a single $\mathcal{O}(n^2)$ relation containing all A, B value pairs that satisfy the predicate. A more efficient representation as a join of two relations can lower the size to $\mathcal{O}(n \log n)$ [8].

F. Relationship to Other Problems

To conclude the tutorial, we discuss related problems that do not fit under the general framework of ranked enumeration. One such problem is *selection* [54]: instead of enumerating the answers to a query in order until some position (say, the median), is it possible to directly "jump" to that answer efficiently? And what changes if, akin to the enumeration framework, we require multiple such accesses after a pre-processing phase [55]? Perhaps surprisingly, these kinds of problems had not been explored until recently and many questions remain open.

Another direction that has been heavily studied is that of *top-k*. Existing work on top-k for joins [4], [56] adopted a cost model that does not penalize large intermediate results. In our previous tutorial [10], we extensively compared the implications of such a model on the design and performance of algorithms. Finally, top-k techniques for single-table queries [57] are different than the topic of this tutorial. These often have a geometric nature [58] and rely on early pruning.

IV. Presenters

Nikolaos Tziavelis is a PhD candidate at Khoury College of Computer Sciences of Northeastern University. His research aims to extend database technology with improved algorithms that achieve non-trivial guarantees. He received a MEng in

Electrical and Computer Engineering from the National Technical University of Athens.

Wolfgang Gatterbauer is an Associate Professor at Khoury College of Computer Sciences at Northeastern University. His research aims to develop scalable algorithms that can leverage structure in data. He received his PhD in Computer Science from Vienna University of Technology, and held positions as PostDoc at University of Washington and Assistant Professor at Carnegie Mellon's Tepper School of Business.

Mirek Riedewald is an Associate Professor at Khoury College of Computer Sciences at Northeastern University. He received his PhD from the University of California at Santa Barbara and held positions as Research Associate at Cornell University as well as visiting positions at Microsoft Research in Redmond and at the Max Planck Institute for Informatics (MPI-I) in Germany. His research is focused on distributed data-intensive computations and algorithms and systems that scale in the size, dimensionality, and arrival speed of data.

References

[1] G. Bagan, A. Durand, and E. Grandjean, "On acyclic conjunctive queries and constant delay enumeration," in *International Workshop on Computer Science Logic (CSL)*, 2007, pp. 208–222.

[2] D. Olteanu and M. Schleich, "Factorized databases," *SIGMOD Record*, vol. 45, no. 2, 2016.

[3] N. Tziavelis, D. Ajwani, W. Gatterbauer, M. Riedewald, and X. Yang, "Optimal algorithms for ranked enumeration of answers to full conjunctive queries," *PVLDB*, vol. 13, no. 9, pp. 1582–1597, 2020.

[4] I. F. Ilyas, G. Beskales, and M. A. Soliman, "A survey of top-k query processing techniques in relational database systems," *ACM Computing Surveys*, vol. 40, no. 4, p. 11, 2008.

[5] M. Yannakakis, "Algorithms for acyclic database schemes," in *VLDB*, 1981, pp. 82–94.

[6] H. Q. Ngo, E. Porat, C. Ré, and A. Rudra, "Worst-case optimal join algorithms," *J. ACM*, vol. 65, no. 3, p. 16, 2018.

[7] C. Berkholz, F. Gerhardt, and N. Schweikardt, "Constant delay enumeration for conjunctive queries: A tutorial," *ACM SIGLOG News*, vol. 7, no. 1, pp. 4–33, 2020.

[8] N. Tziavelis, W. Gatterbauer, and M. Riedewald, "Beyond equi-joins: Ranking, enumeration and factorization," *Proc. VLDB Endow.*, vol. 14, no. 11, pp. 2599–2612, 2021.

[9] R. Bellman, *The theory of dynamic programming*, 1954, vol. 60, no. 6.

[10] N. Tziavelis, W. Gatterbauer, and M. Riedewald, "Optimal join algorithms meet top-k," in *SIGMOD*, 2020, pp. 2659–2665.

[11] R. Bellman, "On a routing problem," *Quarterly of Applied Mathematics*, vol. 16, pp. 87–90, 1958.

[12] S. Bistarelli, U. Montanari, and F. Rossi, "Semiring-based constraint satisfaction and optimization," *J. ACM*, vol. 44, no. 2, p. 201–236, 1997.

[13] M. Abo Khamis, H. Q. Ngo, and A. Rudra, "Faq: questions asked frequently," in *PODS*, 2016, pp. 13–28.

[14] U. Bertele and F. Brioschi, *Nonserial dynamic programming*. Academic Press, 1972.

[15] S. Aji and R. McEliece, "The generalized distributive law," *IEEE Transactions on Information Theory*, vol. 46, no. 2, pp. 325–343, 2000.

[16] M. Mohri, "Semiring frameworks and algorithms for shortest-distance problems," *J. Autom. Lang. Comb.*, vol. 7, no. 3, pp. 321–350, 2002.

[17] T. J. Green, G. Karvounarakis, and V. Tannen, "Provenance semirings," in *PODS*, 2007, p. 31–40.

[18] M. Gondran and M. Minoux, *Graphs, Dioids and Semirings: New Models and Algorithms (Operations Research/Computer Science Interfaces Series)*. Springer, 2008.

[19] P. A. Bernstein and D. W. Chiu, "Using semi-joins to solve relational queries," *J. ACM*, vol. 28, no. 1, pp. 25–40, 1981.

[20] N. Carmeli and M. Kröll, "On the enumeration complexity of unions of conjunctive queries," in *PODS*, 2019, pp. 134–148.

[21] C. Berkholz, J. Keppeler, and N. Schweikardt, "Answering conjunctive queries under updates," in *PODS*, 2017, pp. 303–318.

[22] M. Idris, M. Ugarte, S. Vansummeren, H. Voigt, and W. Lehner, "Efficient query processing for dynamically changing datasets," *SIGMOD Record*, vol. 48, no. 1, pp. 33–40, 2019.

[23] N. Carmeli and M. Kröll, "Enumeration complexity of conjunctive queries with functional dependencies," *Theory Comput. Syst.*, vol. 64, no. 5, pp. 828–860, 2020.

[24] G. Gottlob, G. Greco, N. Leone, and F. Scarcello, "Hypertree decompositions: Questions and answers," in *PODS*, 2016, pp. 57–74.

[25] N. Robertson and P. Seymour, "Graph minors. ii. algorithmic aspects of tree-width," *Journal of Algorithms*, vol. 7, no. 3, pp. 309 – 322, 1986.

[26] G. Gottlob, N. Leone, and F. Scarcello, "Hypertree decompositions and tractable queries," *Journal of Computer and System Sciences*, vol. 64, no. 3, pp. 579 – 627, 2002.

[27] ——, "Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width," *Journal of Computer and System Sciences*, vol. 66, no. 4, pp. 775–808, 2003.

[28] G. Gottlob, Z. Miklós, and T. Schwentick, "Generalized hypertree decompositions: NP-hardness and tractable variants," *J. ACM*, vol. 56, no. 6, p. 30, 2009.

[29] G. Greco and F. Scarcello, "Greedy strategies and larger islands of tractability for conjunctive queries and constraint satisfaction problems," *Inf. Comput.*, vol. 252, pp. 201–220, 2017.

[30] ——, "The power of local consistency in conjunctive queries and constraint satisfaction problems," *SIAM Journal on Computing*, vol. 46, no. 3, pp. 1111–1145, 2017.

[31] M. Grohe and D. Marx, "Constraint solving via fractional edge covers," *ACM TALG*, vol. 11, no. 1, p. 4, 2014.

[32] D. Marx, "Tractable hypergraph properties for constraint satisfaction and conjunctive queries," *J. ACM*, vol. 60, no. 6, pp. 42:1–42:51, 2013.

[33] H. Q. Ngo, "Worst-case optimal join algorithms: Techniques, results, and open problems," in *PODS*, 2018, pp. 111–124.

[34] A. Atserias, M. Grohe, and D. Marx, "Size bounds and query plans for relational joins," *SIAM Journal on Computing*, vol. 42, no. 4, pp. 1737–1767, 2013.

[35] G. Navarro, J. L. Reutter, and J. Rojas-Ledesma, "Optimal Joins Using Compact Data Structures," in *ICDT*, vol. 155, 2020, pp. 21:1–21:21.

[36] H. Q. Ngo, C. Ré, and A. Rudra, "Skew strikes back: New developments in the theory of join algorithms," *SIGMOD Record*, vol. 42, no. 4, pp. 5–16, Feb. 2014.

[37] T. L. Veldhuizen, "Triejoin: A simple, worst-case optimal join algorithm," in *ICDT*, 2014, pp. 96–106.

[38] P. Bourhis, A. Grez, L. Jachiet, and C. Riveros, "Ranked enumeration of MSO logic on words," *CoRR*, vol. abs/2010.08042, 2020.

[39] S. Deep and P. Koutris, "Ranked enumeration of conjunctive query results," in *ICDT*, vol. 186, 2021, pp. 5:1–5:19.

[40] B. Kimelfeld and Y. Sagiv, "Incrementally computing ordered answers of acyclic conjunctive queries," in *International Workshop on Next Generation Information Technologies and Systems (NGITS)*, 2006, pp. 141–152.

[41] X. Yang, D. Ajwani, W. Gatterbauer, P. K. Nicholson, M. Riedewald, and A. Sala, "Any-k: Anytime top-k tree pattern retrieval in labeled graphs," in *WWW*, 2018, pp. 489–498.

[42] X. Yang, M. Riedewald, R. Li, and W. Gatterbauer, "Any-k algorithms for exploratory analysis with conjunctive queries," in *International Workshop on Exploratory Search in Databases and the Web (ExploreDB)*, 2018, pp. 1–3.

[43] M. Ding, S. Chen, N. Makrynioti, and S. Manegold, "Progressive join algorithms considering user preference," in *CIDR*, 2021.

[44] S. Zilberstein, "Using anytime algorithms in intelligent systems," *AI Magazine*, vol. 17, no. 3, pp. 73–83, 1996.

[45] E. L. Lawler, "A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem," *Management science*, vol. 18, no. 7, pp. 401–405, 1972.

[46] K. G. Murty, "An algorithm for ranking all the assignments in order of increasing cost," *Operations Research*, vol. 16, no. 3, pp. 682–687, 1968.

[47] L. Chang, X. Lin, W. Zhang, J. X. Yu, Y. Zhang, and L. Qin, "Optimal enumeration: Efficient top-k tree matching," *PVLDB*, vol. 8, no. 5, pp. 533–544, 2015.

[48] S. E. Dreyfus, "An appraisal of some shortest-path algorithms," *Operations research*, vol. 17, no. 3, pp. 395–412, 1969.

[49] V. M. Jiménez and A. Marzal, "Computing the K shortest paths: A new algorithm and an experimental comparison," in *International Workshop on Algorithm Engineering (WAE)*, 1999, pp. 15–29.

[50] N. Tziavelis, D. Ajwani, W. Gatterbauer, M. Riedewald, and X. Yang, "Optimal algorithms for ranked enumeration of answers to full conjunctive queries," *CoRR*, vol. abs/1911.05582, 2019.

[51] D. Olteanu and J. Závodnỳ, "Size bounds for factorised representations of query results," *TODS*, vol. 40, no. 1, p. 2, 2015.

[52] M. Abo Khamis, R. R. Curtin, B. Moseley, H. Q. Ngo, X. Nguyen, D. Olteanu, and M. Schleich, "On functional aggregate queries with additive inequalities," in *PODS*, 2019, pp. 414–431.

[53] M. A. Khamis, H. Q. Ngo, D. Olteanu, and D. Suciu, "Boolean tensor decomposition for conjunctive queries with negation," in *ICDT*, 2019, pp. 21:1–21:19.

[54] N. Carmeli, N. Tziavelis, W. Gatterbauer, B. Kimelfeld, and M. Riedewald, "Tractable orders for direct access to ranked answers of conjunctive queries," in *PODS*, 2021, pp. 325–341.

[55] N. Carmeli, S. Zeevi, C. Berkholz, B. Kimelfeld, and N. Schweikardt, "Answering (unions of) conjunctive queries using random access and random-order enumeration," in *PODS*, 2020, pp. 393–409.

[56] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," *Journal of Computer and System Sciences*, vol. 66, no. 4, pp. 614–656, 2003.

[57] S. Rahul and Y. Tao, "A guide to designing top-k indexes," *SIGMOD Record*, vol. 48, no. 2, 2019.

[58] K. Mouratidis, "Geometric approaches for top-k queries," *PVLDB.*, vol. 10, no. 12, pp. 1985–1987, 2017.