Optimal Join Algorithms Meet Top-*k*

Nikolaos Tziavelis
Northeastern University
Boston, Massachusetts, USA
ntziavelis@ccs.neu.edu

Wolfgang Gatterbauer
Northeastern University
Boston, Massachusetts, USA
w.gatterbauer@northeastern.edu

Mirek Riedewald
Northeastern University
Boston, Massachusetts, USA
m.riedewald@northeastern.edu

ABSTRACT

Top-k queries have been studied intensively in the database community and they are an important means to reduce query cost when only the "best" or "most interesting" results are needed instead of the full output. While some optimality results exist, e.g., the famous Threshold Algorithm, they hold only in a fairly limited model of computation that does not account for the cost incurred by large intermediate results and hence is not aligned with typical database-optimizer cost models. On the other hand, the idea of avoiding large intermediate results is arguably the main goal of recent work on *optimal join algorithms*, which uses the standard RAM model of computation to determine algorithm complexity. This research has created a lot of excitement due to its promise of reducing the time complexity of join queries with cycles, but it has mostly focused on full-output computation. We argue that the two areas can and should be studied from a unified point of view in order to achieve optimality in the common model of computation for a very general class of top-*k*-style join queries. This tutorial has two main objectives. First, we will explore and contrast the main assumptions, concepts, and algorithmic achievements of the two research areas. Second, we will cover recent, as well as some older, approaches that emerged at the intersection to support efficient *ranked enumeration of join-query results*. These are related to classic work on *k*-shortest path algorithms and more general optimization problems, some of which dates back to the 1950s. We demonstrate that this line of research warrants renewed attention in the challenging context of ranked enumeration for general join queries.

ACM Reference Format:
Nikolaos Tziavelis, Wolfgang Gatterbauer, and Mirek Riedewald. 2020. Optimal Join Algorithms Meet Top-*k*. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3318464.3383132

SIGMOD'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3383132

1 INTRODUCTION

Join-query evaluation is a fundamental problem in databases, hence it is not surprising that recent work on *worst-case-optimal* (WCO) join algorithms [76, 77] generated a lot of excitement. The basic insight is that standard join algorithms that treat multiway joins with cycles as a sequence of pairwise joins are provably suboptimal in that they may produce intermediate results that are asymptotically larger than the largest output this query may produce over *any possible input instance*. By taking a "holistic" approach, WCO join algorithms guarantee a running-time complexity that matches the worst-case output size of a given query [77]. Interestingly, recent work on factorized databases [82] and "optimal" join algorithms [6][1] has shown that the same and even better time complexity can be achieved by decomposing a cyclic join query into multiple acyclic join plans and routing different subsets of the input to different plans. A key insight is that WCO join algorithms are *not output sensitive*: their complexity guarantees do not improve when a query has only a small output, e.g., when none of the input tuples in a given database instance happen to form a result. Similarly, the time-complexity guarantees of WCO join algorithms are weak in the presence of projections (e.g., for Boolean join queries, which ask if the join has any result).

Since worst-case optimality is defined with respect to the largest output of the query over all possible inputs, it is not a natural fit for top-*k* queries, which aim to reduce query cost when only few results are needed. Consider a graph with weighted edges, where lower weights represent greater importance, and the problem of finding the top-*k* *lightest 4-cycles*, i.e., the *k* most important cycles consisting of 4 edges. This, as well as any other graph-pattern query, can be expressed with self-joins of the edge set: here a 4-way join with equality conditions on the endpoints of adjacent edges.[2] Abstractly, all results are sorted according to a *ranking function* and the query needs to return only the first *k* of them. In a graph with n edges, there can be $O(n^2)$ 4-cycles,

[1] We will elaborate more on the distinction between optimal and worst-case-optimal join algorithms in Section 3.
[2] For simplicity we ignore the issue of degenerate cycles, i.e., the same node or edge can appear more than once in the cycle.

therefore a WCO join algorithm would run in time $O(n^2)$. On the other hand, it has been shown that the corresponding Boolean query ("Is there any 4-cycle?") can be answered in $O(n^{1.5})$ [6]. It is tempting to assume that for small k, finding the k lightest cycles will have complexity close to the Boolean query, and as we will demonstrate this turns out to be correct [90].

Interestingly, the above question had not been addressed by the extensive literature on top-k queries in the database context [50, 83]. There exist approaches with optimality guarantees, e.g., the Threshold Algorithm [30], but their optimality holds only in a restricted model of computation where cost is measured in terms of the number of tuples accessed, while the actual computation is essentially "free."[3] We instead will discuss and analyze all top-k algorithms from the point of view of the standard *RAM model* of computation that charges $O(1)$ for each memory access, i.e., it also accounts for cost incurred by large intermediate results and agrees with the model used in the context of (worst-case) optimal join algorithms.

This tutorial will generally survey these two seemingly different areas—optimal joins and top-k—from a unified point of view. We intend to achieve this by highlighting the underlying assumptions made by illustrating important achievements and algorithmic ideas in the two lines of work. By formally defining the common foundations, we are able to reveal fruitful research directions at the intersection: How can we extend optimal join algorithms with ideas from top-k query processing to create frameworks for *optimal ranked enumeration* over general join queries? What types of ranking functions can be supported efficiently? And how can sorting be pushed deep into the join computation? While some recent work has started to explore those questions [16, 21, 61, 90, 93, 94], much is still left to be done.

Audience. The tutorial targets researchers and practitioners who desire an intuitive introduction to recent developments in the theory of optimal join algorithms, including topics such as generalized and fractional hypertree decompositions of cyclic queries, different notions of query width, fractional edge cover, factorized representation, increasingly tight notions of optimality, and enumeration algorithms for join queries. It is also suitable for those interested in a concise comparison of major top-k approaches that were proposed in the context of join queries.

Prerequisites. To make all material accessible to those interested in the practical impact of the techniques, the tutorial will heavily favor intuitive examples and explanations over low-level technical details. In the same spirit, and in line with much of the recent work on optimal join algorithms, we generally take a database-centric view and will present asymptotic complexity results in terms of *data complexity* in $\tilde{O}$-notation (read as "*soft-O*"). Data complexity treats query size (i.e., the size of the query expression itself) as a constant and focuses on scalability in the size of the data. The $\tilde{O}$-notation abstracts away poly-logarithmic factors in input size as those factors often clutter a formula and poly-log grows asymptotically slower than a linear function (hence those factors are considered small compared to even just reading the input once). For instance, consider the following case. Let $f()$ denote some arbitrary computable function, Q the query, n the size of its largest input relation and r the size of its output. Then, a detailed complexity formula such as $O(f(|Q|) \cdot n^{f(|Q|)} + (\log n)^{f(|Q|)} \cdot r)$ would simplify to $\tilde{O}(n^{f(|Q|)} + r)$. Note how the exponent that depends on $|Q|$ does not disappear in the first term and how the entire poly-log factor disappears in the second. Whenever we want to analyze performance differences at finer granularity, we will also show the detailed complexity formulas in standard O-notation.

All material will be self-contained, i.e., we only assume familiarity with fundamental database concepts that would be covered in a typical undergraduate database course, and we do not require previous knowledge of optimal join algorithms or top-k queries.

Outline of the tutorial. This is a 90-minute tutorial consisting of three main parts:

(1) Top-k algorithms for join queries
(2) (Worst-case) optimal join algorithms
(3) Ranked enumeration over join queries: optimality, ranking functions, and empirical comparison of the most promising approaches.

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.[4]

2 PART 1: TOP-K ALGORITHMS

The first part of the tutorial presents core techniques for answering top-k queries in databases with a particular focus on those supporting joins [50, 64], while only briefly touching on top-k problems in other contexts such as single-table queries [83] that often have a geometric nature [68]. In general, top-k aims to prioritize input tuples that could contribute to any of the k top-ranked results over those that cannot, often pruning the latter as early as possible. Complications arise because the importance of a result tuple (often captured by its aggregate weight) typically depends on the weights of the input tuples that join to produce it. This limits

[3] The original motivation for this model are middleware settings where the algorithm is charged for requests made to external input sources.

[4] https://northeastern-datalab.github.io/topk-join-tutorial/

the choices of *ranking functions* for which efficient computation and effective pruning are possible.

One of the best-known top-k approaches is the Threshold Algorithm (TA) [30], for which Fagin, Lotem and Naor received the 2014 Gödel Prize, both for the algorithm's simplicity and its strong instance-optimality guarantee. Conceptually, TA operates on a single table that was partitioned vertically, with each partition being managed by a different external service that knows the ranking only for its partition. A middleware's challenge is then to combine those individual rankings to find the global winners for the full table. TA's cost is measured in term of the number of tuple fragments retrieved by the middleware from the external sources, but it *does not take the actual join cost into account.* This is acceptable in the target application, because TA supports only a very limited type of join, also termed "top-k selection query" [50], where tuple fragments from different partitions join 1-to-1 on a unique object identifier to piece together a row of the full table.

TA marks the culmination of a series of papers where Fagin introduces the problem and proposes an algorithm, now known as "Fagin's algorithm" (FA) [27–29], which does not have TA's strong optimality guarantees. FA also motivated several approaches that essentially proposed TA before the famous TA paper, but without identifying and proving the algorithm's instance optimality. This includes work by Nepal and Ramakrishna [73] and Güntzer et al. [43], the latter of which also incorporates heuristics for deciding which list to fetch tuples from. TA in turn motivated various extensions of the idea to more general join problems, including J* [71], Rank-Join [48], LARA-J* [65], a-FRPA [31], SMART [92], among others surveyed by Ilyas et al [50]. All these algorithms register significant performance gains when the k top-ranked join results depend on only "a few" top-ranked tuples from the input tables. In general, they attempt to minimize how deep down the list they have to go in each pre-sorted input table until they can guarantee that the correct k results have been determined. To achieve the latter, they derive a bound on the score of possible join results containing yet-unseen tuples and update this bound after accessing an input tuple. Our intention here is to highlight the specific innovations introduced by each algorithm, which mostly aim to navigate the tradeoff between cost for accessing tuples [49, 88] and computing improved bounds [87] for early termination.

Similar to TA, analytical results in this space are generally stated in terms of the number of input tuples accessed. We revisit those results and analyze the algorithms in the standard RAM model of computation. We are particularly interested in their worst-case behavior when some of the input tuples contributing to the top-ranked result are at the bottom of an individual input relation and will generally explore to what degree they suffer from large intermediate results, especially for cyclic joins.

3 PART 2: OPTIMAL JOIN ALGORITHMS

The second part of the tutorial presents both classic and state-of-the-art results on optimal processing of join queries, using minimal examples such as path, triangle, and 4-cycle queries in graphs. We provide a brief summary of selected approaches that will be discussed. In addition to algorithms, we will also take a closer look at various competing notions of *optimality* [6, 77]. For the following discussion, recall that n is the size of the largest input relation, r is the size of the output and we generally express complexity results in terms of data complexity in $\tilde{O}$-notation. Furthermore, the cost analysis makes no assumption about the existence of pre-computed data structures on the input relations at query-submission time, including any type of indexes or materialized views. If the algorithm needs such a data structure, it has to create it from scratch, i.e., this cost is reflected in the query time.

For a lower bound, notice that query Q has to examine each input tuple at least once and has to write out each result. This means that join evaluation has complexity at least

$$\Omega(n + r).$$

Somewhat amazingly, the *Yannakakis algorithm* [95] achieves

$$\tilde{O}(n + r)$$

for *acyclic* queries, essentially matching the lower bound.

Its secret of success is the property that after a *full reducer* pass, consisting of semi-join reductions [13] between pairs of joining input relations, the database is left in a state of global consistency [19] , where any intermediate join result can be extended to a valid output tuple.

Unfortunately, as Ngo et al. [77] convincingly argue, for join queries with *cycles* the $\tilde{O}(n + r)$ bound is unattainable based on well-accepted complexity-theoretic assumptions. They therefore propose the notion of *worst-case-optimal (WCO)* join algorithms of time complexity

$$\tilde{O}(n + r_{\mathrm{WC}}),$$

where r_{WC} denotes the size of the greatest possible output of query Q *over any database instance.*

For r_{WC}, Atserias, Grohe, and Marx [7] provide a tight upper bound by connecting join-output size to the *fractional edge cover* of the corresponding query hypergraph. This is now known as the AGM bound, and it is tight in the sense that there exist database instances for which the output size indeed matches the bound. Follow-up work extended the AGM bound to general conjunctive queries with projections and/or functional dependencies [35] as well as degree constraints [5, 6], which generalize the concept of functional dependencies.

A variety of WCO join algorithms have been proposed to match the AGM bound [54, 72, 77, 78, 91]. In contrast to the common "two-relations-at-a-time" approach, i.e., binary join plans, favored by database optimizers, they take a more "holistic" approach by computing a multiway join directly. Consider the often used *triangle query*, a natural join over input relations $R(A,B) = S(B,C) = T(C,A) = \{(1,1),(2,1),\ldots,(n/2,1),(1,2),(1,3),\ldots,(1,n/2)\}$. No matter the join order for a binary join plan,[5] the first binary join produces $O(n^2)$ intermediate results, even though the AGM bound shows that final output size cannot exceed $n^{1.5}$. As a consequence, the binary-join approach has complexity $\tilde{O}(n^2)$, while a WCO join algorithm like Generic-Join [78] or NPRR [77] computes the output in time $\tilde{O}(n^{1.5})$.

Unfortunately, WCO join algorithms are not *output-sensitive*, i.e., their complexity does not improve for database instances resulting in small output. Consider again the triangle query. If there are indeed $\Theta(n^{1.5})$ results, then $\tilde{O}(n^{1.5})$ join time is the best one can hope for. On the other hand, if there are zero triangles for a given database instance, then one would hope to be able to achieve running time closer to $\tilde{O}(n)$, i.e., the time it takes to read the input. This applies also to the Boolean version of the query, which asks *if* there are any triangles, but does not need to return any of them. These issues are addressed by a different notion of optimality that requires the join algorithm to have time complexity

$$\tilde{O}(n^d + r)$$

for the smallest value of parameter d possible [6]. In contrast to WCO join algorithms, this complexity depends on the output size on the given database instance, not the largest output over any database instance. Here d is a *width* parameter that captures the "degree of acyclicity" of the join hypergraph. Intuitively, the smallest possible d for a given query establishes its *intrinsic difficulty*. For acyclic queries, $d = 1$ and hence the ideal complexity $\tilde{O}(n + r)$ is achievable with the Yannakis algorithm as we discussed above.

For cyclic queries, the situation is more complicated and different notions of width have been explored [33]. From a practical point of view, algorithms with $\tilde{O}(n^d + r)$ complexity all follow the same high-level approach. They first decompose a cyclic join query into a tree-shaped acyclic join query and materialize the derived relations needed as input for each tree node. Then they run the Yannakakis algorithm on the acyclic join over the derived relations. The total time complexity is generally determined by the size of the largest derived relation. We will survey the different decomposition methods that have been proposed [36–38, 40–42, 67, 84] and highlight their relationships. The current frontier has been established by the *submodular width* [67]. Its key innovation, from a practical point of view, is that it decomposes a cyclic query into a *union of multiple trees*, each one receiving a subset of the input. This enables lower widths compared to decompositions to a single tree. For example, on the 4-cycle query both the WCO Generic-Join [78] and approaches based on single-tree decompositions have complexity $\tilde{O}(n^2)$, the former due to worst-case output size being quadratic in input size and the latter due to the fractional hypertree width being $d = 2$. In contrast, submodular width is 1.5 and hence algorithms like PANDA [6] that rely on decompositions into multiple trees achieve complexity $\tilde{O}(n^{1.5} + r)$, which is better for *small output size* $r = O(n^{1.5})$.

Decomposition techniques for cyclic queries also play a role in *factorised databases*, which aim to reduce query complexity by cleverly representing (intermediate) results in a factorised format [10, 80–82]. We will survey the key insights of this line of work and then conclude this part of the tutorial with an overview of extensions providing support for aggregates [4, 9, 60]. Due to time constraints, we will only provide pointers to other exciting extensions, including those to machine learning [3, 57, 62, 79, 85, 86], degree information [6, 53], inequalities [2], negation [58], result compression [20], dynamic settings [46, 47, 56], and approaches aiming for stronger notions of optimality [59, 75]. It is also worth noting that some of these novel join algorithms have been implemented in prototype systems for graph processing [1, 45, 55]. A historical perspective on WCO join algorithms together with open problems in the area have recently been summarized by Ngo [74].

4 PART 3: RANKED ENUMERATION OVER JOINS ("ANY-K")

The third part of the tutorial focuses on *optimal ranked enumeration* over both acyclic and cyclic joins, which has started to attract attention recently [16, 21, 61, 90, 93, 94]. A ranked-enumeration algorithm returns the join results in the order of importance as imposed by a ranking function. Its goal is to minimize the time for returning the k top-ranked results *for every value of* k. Stated differently, the algorithm must return query results one-by-one in ranking order without knowing the value of k in advance. While some top-k approaches support this functionality or can easily be extended to do so, others rely on knowing k for pruning lower-ranked results. In order to more clearly distinguish between them, we will refer to ranked-enumeration algorithms also as "*any-k*" join algorithms as a shorthand for "*anytime top-k*."

Despite being reminiscent of the general concept of an anytime algorithm [15, 22, 32, 96], any-k algorithms are not approximating the query result [69]. Instead, they reside squarely at the intersection of top-k and optimal joins, and we will discuss how they are impacted by ideas from both.

[5] The three possible join orders are: $(R \bowtie S) \bowtie T$, $(R \bowtie T) \bowtie S$, or $(S \bowtie T) \bowtie R$.

This tutorial will also highlight an interesting connection to *constant-delay join enumeration* algorithms [8, 12, 24, 89], which produce all query results in quick succession after a short pre-processing phase, albeit in no particular order. Specifically, if an algorithm returns join results with constant delay after spending time t_{prep} on pre-processing, then it guarantees join time $\tilde{O}(t_{\text{prep}}+r)$ and hence gives an output-sensitive complexity guarantee. It therefore would seem natural to extend such approaches to ranked enumeration by investing "a little more" into the pre-processing phase in order to return the results in the right order with constant or logarithmic delay in input size. (The latter is also $\tilde{O}(1)$.)

The center piece of this part of the tutorial are recent results showing that any-k algorithms, for a very general definition of the join query, can be modeled as extensions of non-serial dynamic programming (DP) [90]. This view reveals common foundations between a variety of solutions for problems that had been studied in isolation, often re-inventing the wheel: k-shortest paths [26] and their relationship to DP [14, 17, 18], graph-pattern search [16, 93], and earlier approaches to ranked enumeration over joins [21, 61]. We will demonstrate how these approaches rely on two different major techniques to support the any-k property.

The first is the *Lawler-Murty procedure* [63, 70] that has been used in the database community to design algorithms for ranked enumeration [61] and for graph-pattern search [16, 93]. After identifying the top-ranked result, it cleverly partitions the problem space in order to find the second-best result as the best solution in one of those subspaces; then it recursively proceeds by further partitioning that "winning" subspace. A direct application of the procedure that solves each partition from scratch leads to a delay that is polynomial in the size of the input [61]. Similar attempts with polynomial-delay results have also been made for the equivalent Constraint Satisfaction Problem (CSP) [34, 39]. However, it was recently shown that by exploiting the inherent structure of the join problem, the delay can be reduced to $O(\log k) = \tilde{O}(1)$ [90].

The second approach for adding ranked-enumeration capabilities to a standard DP algorithm originates from k-shortest-path solutions [25, 44, 52, 66] and it relies on a recursive enumeration algorithm that exploits a generalization of the DP principle of optimality [11, 23, 51]. The same recursive call structure appears to have been rediscovered in recent work on ranked enumeration for conjunctive join queries [21].

We will present recent theoretical and empirical evidence [90] that neither of the two major approaches (Lawler-Murty vs. recursive enumeration) dominates the other. In general, these deeper relationships between seemingly different problems and algorithms are fascinating in their own right. Besides, we argue that they are essential for the design of optimal ranked-enumeration algorithms over joins, including generalizations that go beyond natural and Boolean conjunctive queries.

We conclude with an overview of interesting open problems at the intersection of joins and top-k queries.

5 AUTHOR INFORMATION

Nikolaos Tziavelis is a PhD student at the Khoury College of Computer Sciences of Northeastern University. His research interests lie in query processing and ranking problems. He holds a MEng in Electrical and Computer Engineering from the National Technical University of Athens.

Wolfgang Gatterbauer is an Associate Professor at the Khoury College of Computer Sciences at Northeastern University. His research interests lie in the intersection of theory and practice of data management with a particular focus on uncertain and inconsistent data. Prior to joining Northeastern, he was an Assistant Professor at Carnegie Mellon's Tepper School of Business, and before that a PostDoc at University of Washington. He received his PhD in Computer Science at Vienna University of Technology.

Mirek Riedewald is an Associate Professor in the 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 research positions at Microsoft Research in Redmond and at the Max Planck Institute for Informatics (MPI-I) in Germany. His research interests are in data management and analytics, with an emphasis on designing scalable distributed analysis techniques for data-driven science. He has collaborated successfully with scientists from many domains, including ornithology, physics, mechanical and aerospace engineering, and astronomy.

6 ACKNOWLEDGEMENTS

This work was supported in part by the National Institutes of Health (NIH) under award number R01 NS091421 and by the National Science Foundation (NSF) under award number CAREER IIS-1762268. The content is solely the responsibility of the authors and does not necessarily represent the official views of NIH or NSF.

REFERENCES

[1] Christopher R Aberger, Andrew Lamb, Susan Tu, Andres Nötzli, Kunle Olukotun, and Christopher Ré. 2017. Emptyheaded: A relational engine for graph processing. *TODS* 42, 4 (2017), 1–44. https://doi.org/10.1145/3129246

[2] Mahmoud Abo Khamis, Ryan R. Curtin, Benjamin Moseley, Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. 2019. On Functional Aggregate Queries with Additive Inequalities. In *PODS*. 414–431. https://doi.org/10.1145/3294052.3319694

[3] Mahmoud Abo Khamis, Hung Q Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. 2018. In-database learning with sparse tensors. In *PODS*. 325–340. https://doi.org/10.1145/3196959.3196960

[4] Mahmoud Abo Khamis, Hung Q Ngo, and Atri Rudra. 2016. FAQ: questions asked frequently. In *PODS*. 13–28. https://doi.org/10.1145/2902251.2902280

[5] Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. 2016. Computing join queries with functional dependencies. In *PODS*. 327–342. https://doi.org/10.1145/2902251.2902289

[6] Mahmoud Abo Khamis, Hung Q Ngo, and Dan Suciu. 2017. What do Shannon-type Inequalities, Submodular Width, and Disjunctive Datalog have to do with one another?. In *PODS*. 429–444. https://doi.org/10.1145/3034786.3056105

[7] Albert Atserias, Martin Grohe, and Dániel Marx. 2013. Size Bounds and Query Plans for Relational Joins. *SIAM J. Comput.* 42, 4 (2013), 1737–1767. https://doi.org/10.1137/110859440

[8] Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. 2007. On acyclic conjunctive queries and constant delay enumeration. In *International Workshop on Computer Science Logic (CSL)*. 208–222. https://doi.org/10.1007/978-3-540-74915-8_18

[9] Nurzhan Bakibayev, Tomáš Kočiský, Dan Olteanu, and Jakub Závodný. 2013. Aggregation and Ordering in Factorised Databases. *PVLDB* 6, 14 (2013), 1990–2001. https://doi.org/10.14778/2556549.2556579

[10] Nurzhan Bakibayev, Dan Olteanu, and Jakub Závodný. 2012. FDB: A Query Engine for Factorised Relational Databases. *PVLDB* 5, 11 (2012), 1232–1243. https://doi.org/10.14778/2350229.2350242

[11] Richard Bellman and Robert Kalaba. 1960. On k th best policies. *J. Soc. Indust. Appl. Math.* 8, 4 (1960), 582–588. https://doi.org/10.1137/0108044

[12] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. 2017. Answering Conjunctive Queries Under Updates. In *PODS*. 303–318. https://doi.org/10.1145/3034786.3034789

[13] Philip A. Bernstein and Dah-Ming W. Chiu. 1981. Using Semi-Joins to Solve Relational Queries. *J. ACM* 28, 1 (1981), 25–40. https://doi.org/10.1145/322234.322238

[14] Dimitri P. Bertsekas. 2005. *Dynamic Programming and Optimal Control* (3rd ed.). Vol. I. Athena Scientific. http://www.athenasc.com/dpbook.html

[15] Mark Boddy. 1991. Anytime Problem Solving Using Dynamic Programming. In *AAAI*. 738–743. http://dl.acm.org/citation.cfm?id=1865756.1865791

[16] Lijun Chang, Xuemin Lin, Wenjie Zhang, Jeffrey Xu Yu, Ying Zhang, and Lu Qin. 2015. Optimal enumeration: Efficient top-k tree matching. *PVLDB* 8, 5 (2015), 533–544. https://doi.org/10.14778/2735479.2735486

[17] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms* (3rd ed.). The MIT Press. https://dl.acm.org/doi/book/10.5555/1614191

[18] Sanjoy Dasgupta, Christos H Papadimitriou, and Umesh Virkumar Vazirani. 2008. *Algorithms*. McGraw-Hill Higher Education. https://dl.acm.org/doi/book/10.5555/1177299

[19] Rina Dechter. 1992. From Local to Global Consistency. *Artif. Intell.* 55, 1 (1992), 87–108. https://doi.org/10.1016/0004-3702(92)90043-W

[20] Shaleen Deep and Paraschos Koutris. 2018. Compressed representations of conjunctive query results. In *PODS*. 307–322. https://doi.org/10.1145/3196959.3196979

[21] Shaleen Deep and Paraschos Koutris. 2019. Ranked Enumeration of Conjunctive Query Results. *CoRR* abs/1902.02698 (2019). http://arxiv.org/abs/1902.02698

[22] Maarten Van den Heuvel, Peter Ivanov, Wolfgang Gatterbauer, Floris Geerts, and Martin Theobald. 2019. Anytime Approximation in Probabilistic Databases via Scaled Dissociations. In *SIGMOD*. 1295–1312. https://doi.org/10.1145/3299869.3319900

[23] Stuart E Dreyfus. 1969. An appraisal of some shortest-path algorithms. *Operations research* 17, 3 (1969), 395–412. https://doi.org/10.1287/opre.17.3.395

[24] Arnaud Durand and Etienne Grandjean. 2007. First-order queries on structures of bounded degree are computable with constant delay. *ACM TCL* 8, 4 (2007), 21. https://doi.org/10.1145/1276920.1276923

[25] David Eppstein. 1998. Finding the k shortest paths. *SIAM J. Comput.* 28, 2 (1998), 652–673. https://doi.org/10.1137/S0097539795290477

[26] David Eppstein. 2016. *k-Best Enumeration*. Springer, Encyclopedia of Algorithms, 1003–1006. https://doi.org/10.1007/978-1-4939-2864-4_733

[27] Ronald Fagin. 1996. Combining fuzzy information from multiple systems. In *PODS*. 216–226. https://doi.org/10.1145/237661.237715

[28] Ronald Fagin. 1998. Fuzzy queries in multimedia database systems. In *PODS*. 1–10. https://doi.org/10.1145/275487.275488

[29] Ronald Fagin. 1999. Combining fuzzy information from multiple systems. *J. Comput. System Sci.* 58, 1 (1999), 83–99. https://doi.org/10.1006/jcss.1998.1600

[30] Ronald Fagin, Amnon Lotem, and Moni Naor. 2003. Optimal aggregation algorithms for middleware. *J. Comput. System Sci.* 66, 4 (2003), 614–656. https://doi.org/10.1016/S0022-0000(03)00026-6

[31] Jonathan Finger and Neoklis Polyzotis. 2009. Robust and efficient algorithms for rank join evaluation. In *SIGMOD*. 415–428. https://doi.org/10.1145/1559845.1559890

[32] Robert Fink, Jiewen Huang, and Dan Olteanu. 2013. Anytime approximation in probabilistic databases. *VLDB J.* 22, 6 (2013), 823–848. https://doi.org/10.1007/s00778-013-0310-5

[33] Georg Gottlob, Gianluigi Greco, Nicola Leone, and Francesco Scarcello. 2016. Hypertree Decompositions: Questions and Answers. In *PODS*. 57–74. https://doi.org/10.1145/2902251.2902309

[34] Georg Gottlob, Gianluigi Greco, and Francesco Scarcello. 2018. Tree projections and constraint optimization problems: Fixed-parameter tractability and parallel algorithms. *J. Comput. System Sci.* 94 (2018), 11–40. https://doi.org/10.1016/j.jcss.2017.11.005

[35] Georg Gottlob, Stephanie Tien Lee, Gregory Valiant, and Paul Valiant. 2012. Size and treewidth bounds for conjunctive queries. *J. ACM* 59, 3 (2012), 1–35. https://doi.org/10.1145/2220357.2220363

[36] Georg Gottlob, Nicola Leone, and Francesco Scarcello. 2002. Hypertree Decompositions and Tractable Queries. *J. Comput. System Sci.* 64, 3 (2002), 579 – 627. https://doi.org/10.1006/jcss.2001.1809

[37] Georg Gottlob, Nicola Leone, and Francesco Scarcello. 2003. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *J. Comput. System Sci.* 66, 4 (2003), 775–808. https://doi.org/10.1145/375551.375579

[38] Georg Gottlob, Zoltán Miklós, and Thomas Schwentick. 2009. Generalized hypertree decompositions: NP-hardness and tractable variants. *J. ACM* 56, 6 (2009), 30. https://doi.org/10.1145/1568318.1568320

[39] Gianluigi Greco and Francesco Scarcello. 2011. Structural Tractability of Constraint Optimization. In *International Conference on Principles and Practice of Constraint Programming (CP)*. 340–355. https://doi.org/10.1007/978-3-642-23786-7_27

[40] Gianluigi Greco and Francesco Scarcello. 2017. Greedy strategies and larger islands of tractability for conjunctive queries and constraint satisfaction problems. *Inf. Comput.* 252 (2017), 201–220. https://doi.org/10.1016/j.ic.2016.11.004

[41] Gianluigi Greco and Francesco Scarcello. 2017. The power of local consistency in conjunctive queries and constraint satisfaction problems. *SIAM J. Comput.* 46, 3 (2017), 1111–1145. https://doi.org/10.1137/16M1090272

[42] Martin Grohe and Dániel Marx. 2014. Constraint solving via fractional edge covers. *ACM TALG* 11, 1 (2014), 4. https://doi.org/10.1145/2636918

[43] Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kießling. 2000. Optimizing multi-feature queries for image databases. In *VLDB*. 419–428. https://dl.acm.org/doi/10.5555/645926.671875

[44] Walter Hoffman and Richard Pavley. 1959. A Method for the Solution of the Nth Best Path Problem. *J. ACM* 6, 4 (1959), 506–514. https://doi.org/10.1145/320998.321004

[45] Aidan Hogan, Cristian Riveros, Carlos Rojas, and Adrián Soto. 2019. A Worst-Case Optimal Join Algorithm for SPARQL. In *ISWC*. 258–275. https://doi.org/10.1007/978-3-030-30793-6_15

[46] Muhammad Idris, Martín Ugarte, Stijn Vansummeren, Hannes Voigt, and Wolfgang Lehner. 2019. Efficient Query Processing for Dynamically Changing Datasets. *SIGMOD Record* 48, 1 (2019), 33–40. https://doi.org/10.1145/3371316.3371325

[47] Muhammad Idris, Martín Ugarte, Stijn Vansummeren, Hannes Voigt, and Wolfgang Lehner. 2020. General dynamic Yannakakis: conjunctive queries with theta joins under updates. *VLDB J.* 29 (2020), 619–653. https://doi.org/10.1007/s00778-019-00590-9

[48] Ihab F Ilyas, Walid G Aref, and Ahmed K Elmagarmid. 2004. Supporting top-k join queries in relational databases. *VLDB J.* 13, 3 (2004), 207–221. https://doi.org/10.1007/s00778-004-0128-2

[49] Ihab F. Ilyas, Walid G. Aref, Ahmed K. Elmagarmid, Hicham G. Elmongui, Rahul Shah, and Jeffrey Scott Vitter. 2006. Adaptive Rank-Aware Query Optimization in Relational Databases. *TODS* 31, 4 (2006), 1257–1304. https://doi.org/10.1145/1189769.1189772

[50] Ihab F Ilyas, George Beskales, and Mohamed A Soliman. 2008. A survey of top-k query processing techniques in relational database systems. *Comput. Surveys* 40, 4 (2008), 11. https://doi.org/10.1145/1391729.1391730

[51] Víctor M Jiménez and Andrés Marzal. 1999. Computing the k shortest paths: A new algorithm and an experimental comparison. In *International Workshop on Algorithm Engineering (WAE)*. Springer, 15–29. https://doi.org/10.1007/3-540-48318-7_4

[52] Víctor M Jiménez and Andrés Marzal. 2003. A lazy version of Eppstein's K shortest paths algorithm. In *International Workshop on Experimental and Efficient Algorithms (WEA)*. Springer, 179–191. https://doi.org/10.1007/3-540-44867-5_14

[53] Manas Joglekar and Christopher Ré. 2018. It's All a Matter of Degree. *Theory of Computing Systems* 62, 4 (2018), 810–853. https://doi.org/10.1007/s00224-017-9811-8

[54] Oren Kalinsky, Yoav Etsion, and Benny Kimelfeld. 2016. Flexible Caching in Trie Joins. In *EDBT*. 282–293. https://doi.org/10.5441/002/edbt.2017.26

[55] Oren Kalinsky, Benny Kimelfeld, and Yoav Etsion. 2020. The TrieJax Architecture: Accelerating Graph Operations Through Relational Joins. In *ASPLOS*. 1217–1231. https://doi.org/10.1145/3373376.3378524

[56] Ahmet Kara, Hung Q. Ngo, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. 2019. Counting Triangles under Updates in Worst-Case Optimal Time. In *ICDT*. 4:1–4:18. https://doi.org/10.4230/LIPIcs.ICDT.2019.4

[57] Mahmoud Abo Khamis, Hung Q Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. 2018. AC/DC: in-database learning thunderstruck. In *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning (DEEM)*. 1–10. https://doi.org/10.1145/3209889.3209896

[58] Mahmoud Abo Khamis, Hung Q. Ngo, Dan Olteanu, and Dan Suciu. 2019. Boolean Tensor Decomposition for Conjunctive Queries with Negation. In *ICDT*. 21:1–21:19. https://doi.org/10.4230/LIPIcs.ICDT.2019.21

[59] Mahmoud Abo Khamis, Hung Q. Ngo, Christopher Ré, and Atri Rudra. 2016. Joins via Geometric Resolutions: Worst Case and Beyond. *TODS* 41, 4, Article 22 (2016), 45 pages. https://doi.org/10.1145/2967101

[60] Mahmoud Abo Khamis, Hung Q Ngo, and Atri Rudra. 2017. Juggling functions inside a database. *SIGMOD Record* 46, 1 (2017), 6–13. https://doi.org/10.1145/3093754.3093757

[61] Benny Kimelfeld and Yehoshua Sagiv. 2006. Incrementally Computing Ordered Answers of Acyclic Conjunctive Queries. In *International Workshop on Next Generation Information Technologies and Systems (NGITS)*. 141–152. https://doi.org/10.1007/11780991_13

[62] Arun Kumar, Jeffrey Naughton, and Jignesh M Patel. 2015. Learning generalized linear models over normalized data. In *SIGMOD*. 1969–1984. https://doi.org/10.1145/2723372.2723713

[63] Eugene L Lawler. 1972. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management science* 18, 7 (1972), 401–405. https://doi.org/10.1287/mnsc.18.7.401

[64] Chunbin Lin, Jiaheng Lu, Zhewei Wei, Jianguo Wang, and Xiaokui Xiao. 2018. Optimal algorithms for selecting top-k combinations of attributes: theory and applications. *VLDB J.* 27, 1 (2018), 27–52. https://doi.org/10.1007/s00778-017-0485-2

[65] Nikos Mamoulis, Man Lung Yiu, Kit Hung Cheng, and David W Cheung. 2007. Efficient top-k aggregation of ranked inputs. *TODS* 32, 3 (2007), 19. https://doi.org/10.1145/1272743.1272749

[66] Ernesto de Queirós Vieira Martins, Marta Madalena Braz Pascoal, and José Luís Esteves Santos. 2001. A new improvement for a K shortest paths algorithm. *Investigação Operacional* 21, 1 (2001), 47–60. http://apdio.pt/documents/10180/15407/IOvol21n1.pdf

[67] Dániel Marx. 2013. Tractable Hypergraph Properties for Constraint Satisfaction and Conjunctive Queries. *J. ACM* 60, 6, Article 42 (2013), 51 pages. https://doi.org/10.1145/2535926

[68] Kyriakos Mouratidis. 2017. Geometric Approaches for Top-k Queries. *PVLDB*. 10, 12 (2017), 1985–1987. https://doi.org/10.14778/3137765.3137826

[69] Barzan Mozafari. 2017. Approximate Query Engines: Commercial Challenges and Research Opportunities. In *SIGMOD*. 521–524. https://doi.org/10.1145/3035918.3056098

[70] Katta G. Murty. 1968. An Algorithm for Ranking all the Assignments in Order of Increasing Cost. *Operations Research* 16, 3 (1968), 682–687. https://doi.org/10.1287/opre.16.3.682

[71] Apostol Natsev, Yuan-Chi Chang, John R Smith, Chung-Sheng Li, and Jeffrey Scott Vitter. 2001. Supporting incremental join queries on ranked inputs. In *VLDB*. 281–290. http://www.vldb.org/conf/2001/P281.pdf

[72] Gonzalo Navarro, Juan L Reutter, and Javiel Rojas-Ledesma. 2020. Optimal Joins using Compact Data Structures. In *ICDT*. https://arxiv.org/abs/1908.01812

[73] Surya Nepal and M. V. Ramakrishna. 1999. Query processing issues in image (multimedia) databases. In *ICDE*. 22–29. https://doi.org/10.1109/ICDE.1999.754894

[74] Hung Q Ngo. 2018. Worst-case optimal join algorithms: Techniques, results, and open problems. In *PODS*. 111–124. https://doi.org/10.1145/3196959.3196990

[75] Hung Q Ngo, Dung T Nguyen, Christopher Re, and Atri Rudra. 2014. Beyond worst-case analysis for joins with minesweeper. In *PODS*. 234–245. https://doi.org/10.1145/2594538.2594547

[76] Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. 2012. Worst-case Optimal Join Algorithms: [Extended Abstract]. In *PODS*. 37–48. https://doi.org/10.1145/2213556.2213565

[77] Hung Q Ngo, Ely Porat, Christopher Ré, and Atri Rudra. 2018. Worst-case optimal join algorithms. *J. ACM* 65, 3 (2018), 16. https://doi.org/10.1145/3180143

[78] Hung Q Ngo, Christopher Ré, and Atri Rudra. 2014. Skew Strikes Back: New Developments in the Theory of Join Algorithms. *SIGMOD Rec.* 42, 4 (Feb. 2014), 5–16. https://doi.org/10.1145/2590989.2590991

[79] Dan Olteanu and Maximilian Schleich. 2016. F: Regression Models over Factorized Views. *PVLDB* 9, 13 (2016), 1573–1576. https://doi.org/10.14778/3007263.3007312

[80] Dan Olteanu and Maximilian Schleich. 2016. Factorized databases. *SIGMOD Record* 45, 2 (2016). https://doi.org/10.1145/3003665.3003667

[81] Dan Olteanu and Jakub Závodnỳ. 2012. Factorised representations of query results: size bounds and readability. In *ICDT*. 285–298. https://doi.org/10.1145/2274576.2274607

[82] Dan Olteanu and Jakub Závodnỳ. 2015. Size bounds for factorised representations of query results. *TODS* 40, 1 (2015), 2. https://doi.org/10.1145/2656335

[83] Saladi Rahul and Yufei Tao. 2019. A Guide to Designing Top-k Indexes. *SIGMOD Record* 48, 2 (2019). https://doi.org/10.1145/3377330.3377332

[84] Neil Robertson and P.D Seymour. 1986. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms* 7, 3 (1986), 309 – 322. https://doi.org/10.1016/0196-6774(86)90023-4

[85] Maximilian Schleich, Dan Olteanu, Mahmoud Abo-Khamis, Hung Q Ngo, and XuanLong Nguyen. 2019. Learning Models over Relational Data: A Brief Tutorial. In *International Conference on Scalable Uncertainty Management (SUM)*. 423–432. https://doi.org/10.1007/978-3-030-35514-2_32

[86] Maximilian Schleich, Dan Olteanu, and Radu Ciucanu. 2016. Learning linear regression models over factorized joins. In *SIGMOD*. 3–18. https://doi.org/10.1145/2882903.2882939

[87] Karl Schnaitter and Neoklis Polyzotis. 2008. Evaluating rank joins with optimal cost. In *PODS*. 43–52. https://doi.org/10.1145/1376916.1376924

[88] Karl Schnaitter, Joshua Spiegel, and Neoklis Polyzotis. 2009. Depth estimation for ranking query optimization. *VLDB J.* 18, 2 (2009), 521–542. https://doi.org/10.1007/s00778-008-0124-z

[89] Luc Segoufin. 2015. Constant Delay Enumeration for Conjunctive Queries. *SIGMOD record* 44, 1 (2015), 10–17. https://doi.org/10.1145/2783888.2783894

[90] Nikolaos Tziavelis, Deepak Ajwani, Wolfgang Gatterbauer, Mirek Riedewald, and Xiaofeng Yang. 2019. Optimal Algorithms for Ranked Enumeration of Answers to Full Conjunctive Queries. *CoRR* abs/1902.02698 (2019). https://arxiv.org/abs/1911.05582

[91] Todd L. Veldhuizen. 2014. Triejoin: A Simple, Worst-Case Optimal Join Algorithm. In *ICDT*. 96–106. https://doi.org/10.5441/002/icdt.2014.13

[92] Minji Wu, Laure Berti-Equille, Amélie Marian, Cecilia M Procopiuc, and Divesh Srivastava. 2010. Processing top-k join queries. *PVLDB* 3, 1 (2010), 860–870. https://doi.org/10.14778/1920841.1920951

[93] Xiaofeng Yang, Deepak Ajwani, Wolfgang Gatterbauer, Patrick K Nicholson, Mirek Riedewald, and Alessandra Sala. 2018. Any-k: Anytime Top-k Tree Pattern Retrieval in Labeled Graphs. In *WWW*. 489–498. https://doi.org/10.1145/3178876.3186115

[94] Xiaofeng Yang, Mirek Riedewald, Rundong Li, and Wolfgang Gatterbauer. 2018. Any-k Algorithms for Exploratory Analysis with Conjunctive Queries. In *International Workshop on Exploratory Search in Databases and the Web (ExploreDB)*. 1–3. https://doi.org/10.1145/3214708.3214711

[95] Mihalis Yannakakis. 1981. Algorithms for Acyclic Database Schemes. In *VLDB*. 82–94. https://dl.acm.org/doi/10.5555/1286831.1286840

[96] Shlomo Zilberstein. 1996. Using Anytime Algorithms in Intelligent Systems. *AI Magazine* 17, 3 (1996), 73–83. http://rbr.cs.umass.edu/shlomo/papers/Zaimag96.html