

SIGMOD Officers, Committees, and Awardees

Chair

Divyakant Agrawal
Department of Computer Science
UC Santa Barbara
Santa Barbara, California
USA
+1 805 893 4385
agrawal <at> cs.ucsb.edu

Vice-Chair

Fatma Ozcan
Systems Research Group
Google
Sunnyvale, California
USA
+1 669 264 9238
Fozcan <at> google.com

Secretary/Treasurer

Rachel Pottinger
Department of Computer Science
University of British Columbia
Vancouver
Canada
+1 604 822 0436
Rap <at> cs.ubc.ca

SIGMOD Executive Committee:

Divyakant Agrawal (Chair), Fatma Ozcan (Vice-chair), Rachel Pottinger (Treasurer), Juliana Freire (Previous SIGMOD Chair), Chris Jermaine (SIGMOD Conference Coordinator and ACM TODS Editor in Chief), Rada Chirkova (SIGMOD Record Editor), Alexandra Meliou (2024 SIGMOD PC co-chair), S Sudarshan (2024 SIGMOD PC co-chair), Floris Geerts (Chair of PODS), Genoveva Vargas Solar (SIGMOD Diversity and Inclusion Coordinator), Sourav S Bhowmick (SIGMOD Ethics)

Advisory Board:

Yannis Ioannidis (Chair), Phil Bernstein, Surajit Chaudhuri, Rakesh Agrawal, Joe Hellerstein, Mike Franklin, Laura Haas, Renee Miller, John Wilkes, Chris Olsten, AnHai Doan, Tamer Özsu, Gerhard Weikum, Stefano Ceri, Beng Chin Ooi, Timos Sellis, Sunita Sarawagi, Stratos Idreos, and Tim Kraska

SIGMOD Information Directors:

Sourav S Bhowmick, Nanyang Technological University
Byron Choi, Hong Kong Baptist University

Associate Information Directors:

Hui Li (SIGMOD Record), Georgia Koutrika (Blogging), Wim Martens (PODS)

SIGMOD Record Editor-in-Chief:

Rada Chirkova, NC State University

SIGMOD Record Associate Editors:

Lyublena Antova, Manos Athanassoulis, Angela Bonifati, Renata Borovica-Gajic, Vanessa Braganholo, Aaron J. Elmore, George Fletcher, Wook-Shin Han, H V Jagadish, Alfons Kemper, Benny Kimelfeld, Samuel Madden, Kyriakos Mouratidis, Tamer Özsu, Kenneth Ross, Yufei Tao, Pinar Tözün, Immanuel Trummer, and Yannis Velegrakis

SIGMOD Conference Coordinator:

Chris Jermaine, Rice University

PODS Executive Committee:

Floris Geerts (chair), Pablo Barcelo, Leonid Libkin, Hung Q. Ngo, Reinhard Pichler, Dan Suciu

Sister Society Liaisons:

Raghu Ramakrishnan (SIGKDD), Yannis Ioannidis (EDBT Endowment), Christian Jensen (IEEE TKDE)

SIGMOD Awards Committee:

H.V. Jagadish (Chair), Sourav S Bhowmick, Angela Bonifati, David Maier, Sayan Ranu, Wang-Chiew Tan

Jim Gray Doctoral Dissertation Award Committee:

Gustavo Alonso (co-chair), Evaggelia Pitoura (co-chair), Angela Bonifati, Zsolt Istevan, Supun Nakandala, Fatma Ozcan, Huanchen Zhang, and Xiaofang Zhou

SIGMOD Edgar F. Codd Innovations Award

For innovative and highly significant contributions of enduring value to the development, understanding, or use of database systems and databases. Recipients of the award are the following:

Michael Stonebraker (1992)	Jim Gray (1993)	Philip Bernstein (1994)
David DeWitt (1995)	C. Mohan (1996)	David Maier (1997)
Serge Abiteboul (1998)	Hector Garcia-Molina (1999)	Rakesh Agrawal (2000)
Rudolf Bayer (2001)	Patricia Selinger (2002)	Don Chamberlin (2003)
Ronald Fagin (2004)	Michael Carey (2005)	Jeffrey D. Ullman (2006)
Jennifer Widom (2007)	Moshe Y. Vardi (2008)	Masaru Kitsuregawa (2009)
Umeshwar Dayal (2010)	Surajit Chaudhuri (2011)	Bruce Lindsay (2012)
Stefano Ceri (2013)	Martin Kersten (2014)	Laura Haas (2015)
Gerhard Weikum (2016)	Goetz Graefe (2017)	Raghu Ramakrishnan (2018)
Anastasia Ailamaki (2019)	Beng Chin Ooi (2020)	Alon Halevy (2021)
Dan Suciu (2022)	Joseph M. Hellerstein (2023)	Samuel Madden (2024)

SIGMOD Systems Award

For technical contributions that have had significant impact on the theory or practice of large-scale data management systems.

Michael Stonebraker and Lawrence Rowe (2015); Martin Kersten (2016); Richard Hipp (2017); Jeff Hammerbacher, Ashish Thusoo, Joydeep Sen Sarma; Christopher Olston, Benjamin Reed, and Utkarsh Srivastava (2018); Xiaofeng Bao, Charlie Bell, Murali Brahmadesam, James Corey, Neal Fachan, Raju Gulabani, Anurag Gupta, Kamal Gupta, James Hamilton, Andy Jassy, Tengiz Kharatishvili, Sailesh Krishnamurthy, Yan Leshinsky, Lon Lundgren, Pradeep Madhavarapu, Sandor Maurice, Grant McAlister, Sam McKelvie, Raman Mittal, Debanjan Saha, Swami Sivasubramanian, Stefano Stefani, and Alex Verbitski (2019); Don Anderson, Keith Bostic, Alan Bram, Grg Burd, Michael Cahill, Ron Cohen, Alex Gorrod, George Feinberg, Mark Hayes, Charles Lamb, Linda Lee, Susan LoVerso, John Merrells, Mike Olson, Carol Sandstrom, Steve Sarette, David Schacter, David Segleau, Mario Seltzer, and Mike Ubell (2020); Michael Blanton, Adam Bolton, Bill Boroski, Joel Brownstein, Robert Brunner, Tamas Budavari, Sam Carliles, Jim Gray, Steve Kent, Peter Kunszt, Gerard Lemson, Nolan Li, Dmitry Medvedev, Jeff Munn, Deoyani Nandreakar-Heinis, Maria Nieto-Santisteban, Wil O'Mullane, Victor Paul, Don Slutz, Alex Szalay, Gyula Szokoly, Manu Taghizadeh-Popp, Jordan Raddick, Bonnie Souter, Ani Thakar, Jan Vandenberg, Benjamin Alan Weaver, Anne-Marie Weijmans, Sue Werner, Brian Yanny, Donald York, and the SDSS collaboration (2021); Michael Armbrust, Tathagata Das, Ankur Dave, Wenchen Fan, Michael J. Franklin, Huaxin Gao, Maxim Gekk, Ali Ghodsi, Joseph Gonzalez, Liang-Chi Hsieh, Dongjoon Hyun, Hyukjin Kwon, Xiao Li, Cheng Lian, Yanbo Liang, Xiangrui Meng, Sean Owen, Josh Rosen, Kousuke Saruta, Scott Shenker, Ion Stoica, Takuya Ueshin, Shivaram Venkataraman, Gengliang Wang, Yuming Wang, Patrick Wendell, Reynold Xin, Takeshi Yamamuro, Kent Yao, Matei Zaharia, Ruifeng Zheng, and Shixiong Zhu (2022); Aljoscha Krettek, Andrey Zagrebin, Anton Kalashnikov, Arvid Heise, Asterios Katsifodimos, Jiangji (Becket) Qin, Benchao Li, Bowen Li, Caizhi Weng, ChengXiang Li, Chesnay Schepler, Chiwan Park, Congxian Qiu, Daniel Warneke, Danny Cranmer, David Anderson, David Morávek, Dawid Wysakowicz, Dian Fu, Dong Lin, Eron Wright, Etienne Chauchot, Fabian Hueske, Fabian Paul, Feng Wang, Gabor Somogyi, Gary Yao, Godfrey He, Greg Hogan, Guowei Ma, Gyula For, Haohui Mai, Henry Saputra, Hequn Cheng, Igal Shilman, Ingo Bürk, Jamie Grier, Jark Wu, Jincheng Sun, Jing Ge, Jing Zhang, Jingsong Lee, Junhan Yang, Konstantin Knauf, Kostas Kloudas, Kostas Tzoumas, Kete (Kurt) Young, Leonard Xu, Lijie Wang, Lincoln Lee, Lungu Andra, Martijn Visser, Marton Balassi, Matthias J. Sax, Matthias Pohl, Matyas Orhidi, Maximilian Michels, Nico Kruber, Niels Basjes, Paris Carbone, Piotr Nowojski, Qingsheng Ren, Robert Metzger, Roman Khachatryan, Rong Rong, Rui Fan, Rui Li, Sebastian Schelter, Seif Haridi, Sergey Nuyanzin, Seth Wiesman, Shaoxuan Wang, Shengkai Fang, Shuyi Chen, Sihua Zhou, Stefan Richter, Stephan Ewen, Theodore Vasiloudis, Thomas Weise, Till Rohrmann, Timo Walther, Tzu-Li (Gordon) Tai, Ufuk Celebi, Vasiliki Kalavri, Volker Markl, Wei Zhong, Weijie Guo, Xiaogang Shi, Xiaowei Jiang, Xingbo Huang, Xingcan Cui, Xintong Song, Yang Wang, Yangze Guo, Yingjie Cao, Yu Li, Yuan Mei, Yun Gao, Yun Tang, Yuxia Luo, Zhijiang Wang, Zhipeng Zhang, Zhu Zhu, Zili Chen (2023); Zhaojing Luo, Beng Chin Ooi, Wei

Wang, Meihui Zhang, Qingchao Cai, Shaofeng Cai, Gang Chen, Tien Tuan Anh Dinh, Jinyang Gao, Qian Lin, Shicong Lin, Kee Yuan Ngiam, Gene Yan Ooi, Moaz Reyad, Kian-Lee Tan, Anthony K. H. Tung, Sheng Wang, Yuncheng Wu, Zhongle Xie, Naili Xing, Rulin Xing, Wanqi Xue, Sai Ho Yeung, James Yip, Lingze Zeng, Zhaoqi Zhang, Kaiping Zheng, Lei Zhu, Ji Wang (2024).

SIGMOD Contributions Award

For significant contributions to the field of database systems through research funding, education, and professional services. Recipients of the award are the following:

Maria Zemankova (1992)	Gio Wiederhold (1995)	Yahiko Kambayashi (1995)
Jeffrey Ullman (1996)	Avi Silberschatz (1997)	Won Kim (1998)
Raghu Ramakrishnan (1999)	Michael Carey (2000)	Laura Haas (2000)
Daniel Rosenkrantz (2001)	Richard Snodgrass (2002)	Michael Ley (2003)
Surajit Chaudhuri (2004)	Hongjun Lu (2005)	Tamer Özsu (2006)
Hans-Jörg Schek (2007)	Klaus R. Dittrich (2008)	Beng Chin Ooi (2009)
David Lomet (2010)	Gerhard Weikum (2011)	Marianne Winslett (2012)
H.V. Jagadish (2013)	Kyu-Young Whang (2014)	Curtis Dyreson (2015)
Samuel Madden (2016)	Yannis E. Ioannidis (2017)	Z. Meral Özsoyoglu (2018)
Ahmed Elmagarmid (2019)	Philippe Bonnet (2020)	Juliana Freire (2020)
Stratos Idreos (2020)	Stefan Manegold (2020)	Ioana Manolescu (2020)
Dennis Shasha (2020)	Divesh Srivastava (2021)	Christian S. Jensen (2022)
K. Selcuk Candan (2023)	Sihem Amer-Yahia (2024)	

SIGMOD Jim Gray Doctoral Dissertation Award

SIGMOD has established the annual SIGMOD Jim Gray Doctoral Dissertation Award to *recognize excellent research by doctoral candidates in the database field.* Recipients of the award are the following:

- **2006 Winner:** Gerome Miklau. *Honorable Mentions:* Marcelo Arenas and Yanlei Diao
- **2007 Winner:** Boon Thau Loo. *Honorable Mentions:* Xifeng Yan and Martin Theobald
- **2008 Winner:** Ariel Fuxman. *Honorable Mentions:* Cong Yu and Nilesh Dalvi
- **2009 Winner:** Daniel Abadi. *Honorable Mentions:* Bee-Chung Chen and Ashwin Machanavajjhala
- **2010 Winner:** Christopher Ré. *Honorable Mentions:* Soumyadeb Mitra and Fabian Suchanek
- **2011 Winner:** Stratos Idreos. *Honorable Mentions:* Todd Green and Karl Schnaitterz
- **2012 Winner:** Ryan Johnson. *Honorable Mention:* Bogdan Alexe
- **2013 Winner:** Sudipto Das, *Honorable Mention:* Herodotos Herodotou and Wenchao Zhou
- **2014 Winners:** Aditya Parameswaran and Andy Pavlo.
- **2015 Winner:** Alexander Thomson. *Honorable Mentions:* Marina Drosou and Karthik Ramachandra
- **2016 Winner:** Paris Koutris. *Honorable Mentions:* Pinar Tozun and Alvin Cheung
- **2017 Winner:** Peter Bailis. *Honorable Mention:* Immanuel Trummer
- **2018 Winner:** Viktor Leis. *Honorable Mention:* Luis Galárraga and Yongjoo Park
- **2019 Winner:** Joy Arulraj. *Honorable Mention:* Bas Ketsman
- **2020 Winner:** Jose Faleiro. *Honorable Mention:* Silu Huang
- **2021 Winner:** Huanchen Zhang, *Honorable Mentions:* Erfan Zamanian, Maximilian Schleich, and Natacha Crooks
- **2022 Winner:** Chenggang Wu, *Honorable Mentions:* Pingcheng Ruan and Kexin Rong
- **2023 Winner:** Supun Nakandala, *Honorable Mentions:* Benjamin Hilprecht and Zongheng Yang
- **2024 Winner:** Daniel Kang, *Honorable Mentions:* Wei Dong, Jialin Ding, and Yisu Remy Wang

A complete list of all SIGMOD Awards is available at: <https://sigmod.org/sigmod-awards/>

[Last updated: June 1, 2024]

Editor's Notes

Welcome to the September 2024 issue of the ACM SIGMOD Record!

This issue starts with the Database Principles column presenting an article by Tziavelis, Gatterbauer, and Riedewald on the topic of ranked enumeration of database queries. The authors cover any-k algorithms for join queries that push ranking into joins, allowing for the generation of the top-ranked answers while avoiding materialization of intermediate results. A key insight of the work is the connection between ranked query enumeration and the fundamental problem of computing the k-th shortest path in a graph. The experimental results show the algorithms to be very efficient in practice. The article would be of interest to the readers who look for database algorithms with solid theoretical guarantees, and in particular to practitioners looking to implement and optimize any-k approaches.

The Research Articles column features an article by Wu and Wang. The article focuses on the problem of cost-based query optimization under the assumption that the unit costs of executing certain types of query-processing operations (such as table scans or joins) are variables, rather than constants. The authors discuss how the problem connects to the hyper-parameter optimization problem in AutoML, and showcase impressive performance results for the queries optimized under the resulting paradigm, both in the single-query and in the multiple-query workload settings. The article concludes with suggestions for future work in this problem space.

The Reminiscences on Influential Papers column presents contributions by Goetz Graefe and Raja Appuswamy.

The DBrainstorming column, whose goal is to discuss new and potentially controversial ideas that might be of interest and benefit to the research community, features an article by Sayan Ranu. The contribution focuses on the problem of lack of interpretability of the predictions provided by graph neural networks (GNNs), with the issues posing significant barriers to the adoption of GNNs in critical domains, including healthcare and finance. The article outlines key challenges in this space and the author's ideas for overcoming them.

The Reports column presents a contribution by Miedema and colleagues, which summarizes the outcomes of the Second International workshop on Data Systems Education: Bridging Education Practice with Education Research (DataEd '23). The workshop was held in conjunction with the SIGMOD '23 conference in Seattle, USA on June 23, 2023. The aim of the workshop was to provide a dedicated venue for presenting and discussing data-management systems education experiences and research, by bringing together the database and computing-education research communities to share findings, to compare and exchange perspectives and methods, and to look for opportunities for mutual progress in data systems education. The program featured two keynote talks, eight research presentations, and a discussion session. The article presents the workshop's main results, observations, and emerging research directions.

On behalf of the SIGMOD Record Editorial board, I hope that you enjoy reading the September 2024 issue of the SIGMOD Record!

Your submissions to the SIGMOD Record are welcome via the submission site:

<https://mc.manuscriptcentral.com/sigmodrecord>

Prior to submission, please read the Editorial Policy on the SIGMOD Record's website:

<https://sigmodrecord.org/sigmod-record-editorial-policy/>

Rada Chirkova

September 2024

Past SIGMOD Record Editors:

Yanlei Diao (2014-2019)	Ioana Manolescu (2009-2013)	Alexandros Labrinidis (2007-2009)
Mario Nascimento (2005-2007)	Ling Liu (2000-2004)	Michael Franklin (1996-2000)
Jennifer Widom (1995-1996)	Arie Segev (1989-1995)	Margaret H. Dunham (1986-1988)
Jon D. Clark (1984-1985)	Thomas J. Cook (1981-1983)	Douglas S. Kerr (1976-1978)
Randall Rustin (1974-1975)	Daniel O'Connell (1971-1973)	Harrison R. Morse (1969)

Ranked Enumeration for Database Queries

Nikolaos Tziavelis^{*}
UC Santa Cruz
ntziavel@ucsc.edu

Wolfgang Gatterbauer
Northeastern University
w.gatterbauer@northeastern.edu

Mirek Riedewald
Northeastern University
m.riedewald@northeastern.edu

ABSTRACT

Ranked enumeration is a query-answering paradigm where the query answers are returned incrementally in order of importance (instead of returning all answers at once). Importance is defined by a ranking function that can be specific to the application, but typically involves either a lexicographic order (e.g., “ORDER BY R.A, S.B” in SQL) or a weighted sum of attributes (e.g., “ORDER BY 3*R.A + 2*S.B”). Recent work has introduced *any-k algorithms* for (multi-way) join queries, which push ranking into joins and avoid materializing intermediate results until necessary. The top-ranked answers are returned asymptotically faster than the common join-then-rank approach of database systems, resulting in orders-of-magnitude speedup in practice.

In addition to their practical usefulness, these techniques complement a long line of theoretical research on *unranked enumeration*, where answers are also returned incrementally, but with no explicit ordering requirement. For a broad class of ranking functions with certain monotonicity properties, including lexicographic orders and sum-based rankings, the ordering requirement surprisingly does not increase the asymptotic time or space complexity, apart from logarithmic factors.

A key insight is the connection between ranked enumeration for database queries and the fundamental task of computing the k^{th} -shortest path in a graph. Although this connection is important for grounding the problem in the literature, it can obfuscate the simplicity of the algorithm. In this article, we adopt a pragmatic approach and present a slightly simplified version of the algorithm without the shortest-path interpretation. We believe that this will benefit practitioners looking to implement and optimize any- k approaches.

1 Introduction

Data analytics queries can generate large intermediate or final results, rendering data systems unresponsive. A primary culprit is the join operator,

^{*}Work done while at Northeastern University.

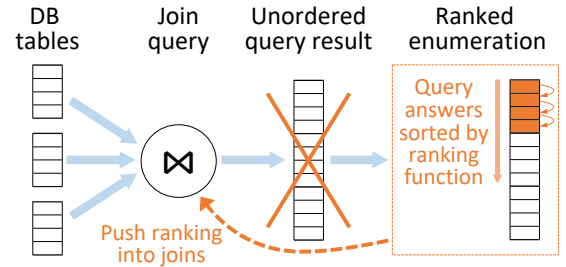


Figure 1: Enumerating the query answers in ranked order without first materializing the unordered query result. *Sorting is pushed into the join operation* so that joining and ranking are interleaved.

which combines data from different tables, potentially causing a combinatorial explosion in the output. Consequently, traditional join-processing techniques can become infeasible, or they simply take too long before delivering any answer to the user or to the next step in a data-processing pipeline. Work on *enumeration* [6, 41] addresses this by returning query answers incrementally as quickly as possible, even when the full query output is too large to compute. However, enumeration traditionally does not support a desired order (or *ranking*) specifying, which answers should be returned first. We thus refer to it as *unranked enumeration*. In practice, certain answers may be preferred over others based on some notion of importance or relevance. For instance, higher importance may be assigned to newer or more trusted data. *Ranked enumeration* [24, 44] therefore augments enumeration with a total-order feature over the query answers, formalized by a ranking function (e.g., expressed by an ORDER BY clause in SQL).

Database systems today follow a join-then-rank approach, i.e. they first compute *all join answers* and then apply the ranking (by sorting either incrementally or in batch). One way to think about the improvement we seek is that we want to “push”

```

SELECT Cit1.PaperID, Cit2.PaperID, Cit3.PaperID,
       Cit3.CitedPaperID, Cit1.InflWeight +
       Cit2.InflWeight + Cit3.InflWeight AS Weight
FROM Cit Cit1, Cit Cit2, Cit Cit3
WHERE Cit1.CitedPaperID = Cit2.PaperID AND
      Cit2.CitedPaperID = Cit3.PaperID
ORDER BY Weight

```

Figure 2: SQL query for ranking chains of highly influential citations.

the ranking operator deeper into the query plan. While this resembles typical database optimizations, such as pushing projections before joins, the task is more challenging, because join and ranking operators generally do not commute. Novel algorithms are required, where joining and ranking are interleaved.¹

Performance Goal. How can performance for such an algorithm be measured? The top-ranked answers should be returned quickly without wasting resources on low-ranked ones, similar to classic top- k queries [29]. However, in contrast to top- k , where “pruning” techniques based on the *given* number of returned answers k can be leveraged,² a ranked-enumeration algorithm does not know the value k in advance. Instead of pruning, it can at best *postpone* work on lower-ranked answers, providing guarantees *no matter how many answers are eventually returned*. We are thus interested in the *Time-To- k* , or $TT(k)$, for any possible value of k . This gave rise to the “any- k ” label, quasi an “any-time top- k ” algorithm [14, 53, 54].

Note that a stricter and popular [6, 28, 38, 41] measure of performance involves combining *preprocessing time* (i.e., $TT(1)$) with the *worst-case delay between answers* (i.e., the maximum inter-arrival time). However, lowering the worst-case delay may have no practical benefit if it does not also improve $TT(k)$ [18, 20, 46]. Adopting $TT(k)$ allows for situations where a spike in delay is offset by shorter delays in *previous* iterations. An established example where this difference occurs is incremental QuickSort [40] which guarantees $TT(k) = \mathcal{O}(n + k \log k)$, but has a linear worst-case delay between answers.

An Example. Consider a bibliography dataset that stores the influence of research papers on later

¹Even simpler top-1 queries are not efficiently supported by current systems. For a minimum example in PostgreSQL, see slide 20: https://northeastern-datalab.github.io/cs7240/sp24/download/cs7240-T3-U1-Acyclic_Queries.pdf.

²Besides the requirement of k being fixed in advance, older work on top- k joins assumes a cost model that accounts for data access, but not for intermediate results [48, Part 1].

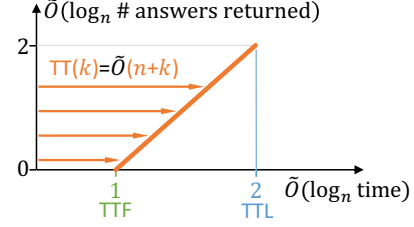


Figure 3: Ranked enumeration guarantees for the query of Figure 2: The first answer (TTF for Time-To-First) is returned in $\tilde{O}(n)$ and the last answer (TTL for Time-To-Last) in $\tilde{O}(n^2)$.

papers that cite them. Each tuple in relation $\text{Cit}(\text{PaperID}, \text{CitedPaperID}, \text{InflWeight})$ states that a paper with ID CitedPaperID influenced a later paper PaperID with a numerical weight InflWeight . For the sake of the example, assume that the influence weight has been precomputed by some prediction technique and takes on an integer value in range $[1, 10]$, with 1 being the most influential. To extract chains of highly influential citations, we can write the join query

$$\text{Cit}(p_1, p_2, s_1), \text{Cit}(p_2, p_3, s_2), \text{Cit}(p_3, p_4, s_3)$$

and order its answers in ascending sequence of the SUM $s_1 + s_2 + s_3$. For readers unfamiliar with Datalog, note that relation Cit appears three times to indicate a self-join (which requires renaming to $\text{Cit1}, \text{Cit2}, \text{Cit3}$ in SQL as shown in Figure 2) and that a variable like p_2 appearing more than once indicates an equi-join between the corresponding columns (i.e., $\text{Cit1}.p_2 = \text{Cit2}.p_2$). How fast can ranked enumeration be here? The entire query output can have size n^2 in the worst case [5]. On the other hand, simply checking if any query answer exists (called the *Boolean* query) takes $\Theta(n)$ [55]. Ranked enumeration aims to cover the continuum between those two with $TT(k) = \tilde{O}(n+k)$, as shown in Figure 3. The \tilde{O} notation abstracts away logarithmic factors in n and k introduced by join indexes or sorting (by $s_1 + s_2 + s_3$).

Prioritizing Computation. To build intuition, let us first consider how unranked enumeration works. If we were to follow a standard table-at-a-time approach, we would start by joining $\text{Cit1} \bowtie \text{Cit2}$. This is a costly bulk computation of time complexity $\tilde{O}(n^2)$. However, it would not yet produce a single query answer because table Cit3 has not been checked. To produce answers as quickly as possible, we need to be more careful in where we spend resources and prioritize differently. Instead of a table-at-a-time, a *tuple-at-a-time* approach is needed. We start with only one tuple from Cit1 , look up the

matches in Cit2, pick one, and then look up the Cit3 matches to produce one answer. This strategy can be implemented using a pipelined execution in a database system. The standard unranked enumeration algorithm [6] achieves $\tilde{O}(n + k)$ by following such an approach, preceded by a $\tilde{O}(n)$ -time semi-join reduction [55], which removes “dangling” tuples that do not contribute to the final output.

Ranked enumeration appears more challenging because additional prioritization is required to avoid low-ranking query answers. Interestingly, a more careful look at the unranked enumeration algorithm [6] reveals that, with appropriate sorting of the input relations, the output naturally follows a *lexicographic order*. A lexicographic order is defined by a sequence of variables, such as $s_1 \rightarrow s_2 \rightarrow s_3$. It means that the answers are first ordered by variable s_1 , then by s_2 , and then by s_3 (ORDER BY Cit1.InflWeight, Cit2.InflWeight, Cit3.InflWeight in SQL). This heavily prioritizes the weight of the first citation in the chain; a chain with weights $1 \rightarrow 10 \rightarrow 10$ would be ranked higher than a chain with weights $2 \rightarrow 1 \rightarrow 1$. The enumeration algorithm of Bagan et al. [6] is capable of producing such an order, granted that we first sort each copy of Citi by InflWeight.

But what if a different order that is “inconvenient” for the algorithm is required? As we will discuss in more detail, certain lexicographic orders, such as $s_2 \rightarrow s_1 \rightarrow s_3$ cannot be achieved by this approach. Moreover, for SUM ranking, the situation is more difficult because a high-ranking tuple in Cit1 might only join with low-ranking tuples in Cit2 and Cit3, leading to low-ranking answers in aggregate. Addressing this requires a stronger form of prioritization that incorporates *lookahead* information about tuples and weights that come later in the query plan.

Any- k Algorithms. Recent developments led to the design and implementation of any- k algorithms achieving $TT(k) = \tilde{O}(n + k)$ for acyclic join queries and appropriately monotone ranking functions [24, 44]. These include *all* lexicographic orders, SUM, as well as MIN and MAX. In our example, the first $k = \mathcal{O}(1)$ answers are obtained after only $\tilde{O}(n)$, and—if the enumeration is carried out to the end—the last answer in $\tilde{O}(n^2)$, matching the join-then-rank approach. Compared to unranked enumeration, ranking by $s_1 + s_2 + s_3$ introduces only a logarithmic factor in k .

Although multiple any- k algorithms exist, their complexity differences concern logarithmic factors and treating query size as a variable that can grow arbitrarily, which may not always materialize in practice. In this article, we cater to practicality and

ease of understanding, focusing on data complexity, guarantees in \tilde{O} without logarithmic factors, and on the easiest-to-understand variant.³ We describe the algorithm in a streamlined way, without the graph abstraction that has been used [44] to highlight the connection to earlier work on shortest-path enumeration [30, 33].

Organization. The rest of this article is organized as follows. Section 2 introduces necessary concepts and notation. Section 3 presents a simple algorithm that works for certain lexicographic orders and explores which lexicographic orders are achievable with this algorithm. Section 4 takes on the harder case of SUM. Section 5 discusses several extensions that generalize the approach to more expressive queries and ranking functions. Section 6 concludes and provides directions for future work.

2 Basic Concepts

We focus on Select-Project-Join queries, which we formally define in the usual way as Conjunctive Queries. Throughout the article, we use $[m]$ to denote the set of integers $\{1, \dots, m\}$.

Database. A *database* D is a set of finite relations $\{R_1, \dots, R_m\}$, where each R_i for $i \in [m]$ has arity $\text{ar}(R)$ (i.e., $\text{ar}(R)$ attributes or columns) and draws values from a fixed infinite domain dom , (i.e., $R_i \subseteq \text{dom}^{\text{ar}(R_i)}$). The size of the database n is the number of tuples across all relations.

Query. In Datalog, a *Conjunctive Query* (CQ) Q is an expression $Q(\mathbf{Y}) :- R_1(\mathbf{V}_1), \dots, R_\ell(\mathbf{V}_\ell)$, where each \mathbf{V}_i for $i \in [\ell]$ is a list of either *variables* (representing database attributes) or constants from dom (encoding selection). Each *atom* $R_i(\mathbf{V}_i)$ refers to a (not necessarily distinct) database relation with $|\mathbf{V}_i|$ attributes. If \mathbf{X} is the set of all distinct variables appearing in all lists \mathbf{V}_i for $i \in [\ell]$, then the variables \mathbf{Y} (representing output attributes) need to be a subset of \mathbf{X} and are called *free*. A *Join Query* (JQ) is a special case of a CQ where all variables are free (i.e., $\mathbf{Y} = \mathbf{X}$). Multiple atoms are allowed to refer to the same relation, resulting in a *self-join*. The query size, measured by the number of symbols in the query, is assumed to be $\mathcal{O}(1)$. This is often referred to as *data complexity* [51] and it is relevant in practice because while new data may be collected, the query size does not typically grow unboundedly.

Queries are evaluated over a database D and produce a result $Q(D)$. A *query answer* or *output tuple* is an element $q \in Q(D)$. The occurrence of the same

³The specific variant we present is ANYK-PART with eager sorting [44, Figure 6].

variable in different atoms encodes an *equi-join* condition, implying equality between the corresponding attributes. A typical preprocessing step for all algorithms is to (1) remove self-joins from the query by copying database tables and (2) remove selections on individual relations (like $R(x, 1)$ or $R(x, x)$) by filtering. These operations take $\mathcal{O}(n)$ and can be ignored because the cost is asymptotically the same as reading the database once. Afterwards, a naive evaluation strategy to compute $Q(D)$ (which helps to understand the query semantics) is to (i) materialize the Cartesian product of the ℓ relations, (ii) select tuples that satisfy the equi-joins, and (iii) project on the \mathbf{Y} attributes.

Acyclicity. A CQ is (alpha-)acyclic [16] if it admits a *join tree*. A join tree is a rooted tree whose nodes are the query atoms and for each variable x , all tree nodes containing x form a connected subtree.⁴ The acyclicity of a CQ can be tested, and a corresponding join tree can be constructed, in linear time in the query size [42].

Ranking. Ranked enumeration assumes a user-specified *ranking function* that orders the query answers $Q(D)$ by mapping them to a domain W equipped with a total order \preceq . Ties are broken arbitrarily. Given a query Q , a *lexicographic order* L is a sequence of query variables $x_1 \rightarrow x_2 \rightarrow \dots$, implying that the answers are first compared by the values of x_1 , and if tied by the values of x_2 , and so on. A *partial* lexicographic order contains a strict subset of the query variables. Another case is *SUM*, given by an expression $f_1(x_1) + f_2(x_2) + \dots$, where f_1, f_2, \dots can be arbitrary, $\mathcal{O}(1)$ -computable functions mapping dom to \mathbb{R} . The ranking may alternatively be defined using values on the database tuples instead of the query variables; the latter can be reduced to the former in linear time as we will see in more detail in Section 4.1.

3 Enumeration by Lexicographic Order

We begin with the lexicographic orders that can be produced as a by-product of the standard “un-ranked” enumeration algorithm through a minor extension (i.e., pre-sorting all input relations according to the lexicographic order). Although various descriptions of this algorithm exist in the literature using different abstractions [6, 11, 38, 41], it is often overlooked that it can easily produce query answers according to certain lexicographic orders.

We offer a detailed description that (i) is easy to implement and (ii) generalizes to SUM (Section 4)

⁴For an illustration, please see <https://www.youtube.com/watch?v=toi7ysuyRkw&t=340> [49].

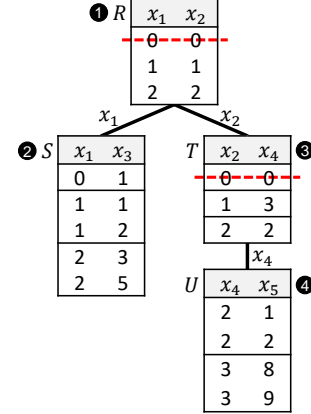


Figure 4: An example database for the join query $R(x_1, x_2), S(x_1, x_3), T(x_2, x_4), U(x_4, x_5)$. The relations are organized in a join tree. Red marks indicate tuples removed by the semijoin reduction. Also shown are shared variables between child-parent pairs and the relation ordering rel used by the lexicographic enumeration algorithm.

and other orders. We focus on acyclic JQs and discuss how this restriction can be lifted in Section 5.

The algorithm consists of two phases. First, the preprocessing phase builds essential data structures such as join indexes and applies a semijoin reduction [55] to remove dangling tuples from the input relations. Then, the enumeration phase traverses the relations using the indexes to connect joining tuples. The $\tilde{\mathcal{O}}(n+k)$ complexity guarantee for $\text{TT}(k)$ hinges on the semijoin filtering, which eliminates “dead-ends” by ensuring that every partial query answer—generated by joining tuples from a subset of relations—can be extended to a complete query answer. We will detail both phases in Sections 3.1 and 3.2, then examine, which lexicographic orders can be supported by this algorithm in Section 3.3.

As a guiding example, we use the query $R(x_1, x_2), S(x_1, x_3), T(x_2, x_4), U(x_4, x_5)$ and show how to achieve the order $x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4 \rightarrow x_5$. An example database is shown in Figure 4.

3.1 Bottom-up Preprocessing Phase

Join Order. The preprocessing phase starts by organizing the relations in a (rooted) join tree \mathcal{T} . Unlike a database query plan, a join tree does not fully specify the join order. It only determines that a parent relation must be processed before its children (also called a topological sort). Hence any order that respects this constraint can be followed by the enumeration algorithm. Let function rel denote such an order, i.e., it maps the integers $[\ell]$ to database relations, where ℓ is the number of rela-

tions. In our example, we have $\text{rel}(1) = R, \text{rel}(2) = S, \text{rel}(3) = T, \text{rel}(4) = U$ (see Fig. 4). This means that relation S will always be visited before relation T during enumeration. To encode the tree structure, we refer to the parent of the r -th relation in the order as $\text{pr}(r)$, for $r \in [2, \ell]$.

Join Indexes. Next, we build join indexes, e.g., B-trees or hash indexes, allowing us to find matching tuples efficiently. We abstract an index as a function $\text{JoinIndex}_{R \rightarrow S}$, which, given a tuple $t \in R$, returns a list M of S tuples that agree with t on the join attributes between R and S (i.e., the common variables between the atoms). To achieve the desired $\text{TT}(k)$ guarantees, the index must be built in $\tilde{O}(n)$ with lookups in $\tilde{O}(1)$ (not including the time it takes to read M). We construct one index for each parent-child pair in the join tree, i.e., $\text{JoinIndex}_{R \rightarrow S}$ based on x_1 , $\text{JoinIndex}_{R \rightarrow T}$ based on x_2 , $\text{JoinIndex}_{T \rightarrow U}$ based on x_4 in our example. Figure 4 shows each relation grouped by the attributes that join with the parent, i.e., the image of $\text{JoinIndex}_{\text{pr}(r) \rightarrow \text{rel}(r)}$ for the r -th relation, $r \in [2, \ell]$. The root has no grouping.

Semijoin reduction. Using the join tree and indexes, we perform a *semijoin reduction* exactly as in the bottom-up step of the Yannakakis algorithm [55]. The relations are traversed in reverse topological order with a semijoin applied for each parent-child pair. In our example, the semijoins are executed in the following order:

$$T = T \ltimes U, \quad R = R \ltimes T, \quad R = R \ltimes S$$

This step is crucial for our desired complexity guarantee. To understand why, consider tuple $R(0, 0)$, for which there are matching tuples in S and T , but none in U . Consequently, the time processing $R(0, 0)$ is wasted, without producing an output tuple. With sufficiently many such “dangling” tuples, the time between consecutive answers would grow to exceed $\text{TT}(k) = \tilde{O}(n + k)$. The semijoin reduction prevents this by removing dangling tuples like $R(0, 0)$ and $T(0, 0)$. Notice that $S(0, 1)$ is dangling, but not removed. Removing all dangling tuples would require a *full reduction* [9, 13], which is not necessary for the enumeration algorithm. It is easy to show that any remaining dangling tuples will never be accessed by top-down traversals.

Algorithm 1 presents the semijoin reduction expressed in a way that easily generalizes to support other orders, as we will see in Section 4. Specifically, it can be viewed as message passing at the tuple level: Each tuple pulls “messages” from joining tuples in the children relations, determines its own state based on the messages, and later passes a mes-

Algorithm 1: Preprocessing for lexicographic enumeration (Section 3.1).

Input: acyclic JQ Q (without self-joins), database D , join tree \mathcal{T} , lexicographic order L , relation ordering rel consistent with \mathcal{T}

```

1 Output: reduced and sorted database  $D'$ ,  $\text{JoinIndex}_{R \rightarrow S}$ 
   for each parent  $R$  and child  $S$  in  $\mathcal{T}$ 
2 Initialize  $\text{val}(t) = \text{True}$  for all tuples  $t$  of all relations
3 //Process relations in reverse  $\text{rel}$  order (bottom-up in  $\mathcal{T}$ )
4 for  $i = \ell$  down to 2 do
5   relation  $S = \text{rel}[i]$ ; relation  $R = \text{pr}(S)$ 
6   //Relation  $S$  has been reduced in a previous iteration
7   (or is a leaf)
8   Construct  $\text{JoinIndex}_{R \rightarrow S}$  on shared attributes
9   Sort  $\text{JoinIndex}_{R \rightarrow S}$  entries by  $L$ 
10  for tuple  $t \in R$  do
11     $M = \text{JoinIndex}_{R \rightarrow S}(t)$ 
12    //Memoization:  $\text{val}(M)$  is reused
13    if  $\text{val}(M)$  not already computed then
14       $\text{val}(M) = \text{False} \vee \bigvee_{t' \in M} \text{val}(t')$ 
15       $\text{val}(t) = \text{val}(t) \wedge \text{val}(M)$ 
16      if not  $\text{val}(t)$  then remove  $t$  from  $R$  in  $D$ 
17 Sort the root  $R$  by  $L$ 
18 return  $D$ ,  $\text{JoinIndex}_{R \rightarrow S}$  for all  $(R, S) \in \mathcal{T}$ 

```

sage up the tree. The “message” here is a Boolean value that indicates whether matching tuples exist in the subtree. If the aggregated message from at least one of the children relations is “False”, then the tuple is removed and a “False” message is propagated upwards. Note that parents are “pulling” instead of children “pushing” messages so that we can use the parent-to-child join indexes that we anyway need in the enumeration phase. The algorithm employs *memoization* for the aggregated message of a join group (Line 13), since multiple tuples in the parent relation may access it. This is important in order to guarantee linear time.

Sorting. When we build a join index, we sort its entries (i.e., the tuples within the same join group) by the same lexicographic order. In the example, the entries of $\text{JoinIndex}_{R \rightarrow S}$ are sorted by $x_1 \rightarrow x_3$, the entries of $\text{JoinIndex}_{R \rightarrow T}$ by $x_2 \rightarrow x_4$, and so on. The join index is built after reducing a relation with messages from its children, and sorted thereafter. The tuples of the root relation are considered to belong to the same join group (as if a parent relation with an empty set of join variables to group-by existed) and are also sorted. Slightly abusing the notation, we treat a relation as a sorted list of tuples; e.g., $R[1]$ denotes the first tuple of R .

3.2 Top-down Enumeration Phase

While the semijoin reduction proceeds bottom-up in the opposite direction of relation order rel (Line 4), the enumeration phase traverses the relations top-down. We start with tuple $R[1] = R(1, 1)$ and, through the join indexes, find the first match

in every relation, yielding the first query answer $(1, 1, 1, 3, 8)$.⁵ In the second iteration, we proceed with the matches from the last relation, i.e., tuple $(3, 9)$ from U , obtaining $(1, 1, 1, 3, 9)$. This exhausts all matches in U , therefore in the third iteration the algorithm backtracks to the next match in preceding relation T . Since no second match exists in T , we backtrack once again to S , encountering $(1, 2)$ there. With $(1, 1, 2)$ as a partial answer, the algorithm proceeds forward to T, U to obtain $(1, 1, 2, 3, 8)$. The process continues analogously, returning answers $(1, 1, 2, 3, 9)$, $(2, 2, 3, 2, 1)$, etc.⁶

This enumeration can be implemented recursively, akin to a standard depth-first search (DFS). Equivalently, we implement it with a *stack* of partial query answers (LIFO), which tracks the current frontier. A partial answer contains matched tuples from only a subset of the relations. When a partial answer is popped from the stack and we extend it into a complete answer, alternatives that use the next available tuple are pushed back onto the stack, starting from the current relation. The extension of a partial answer always selects the first matching tuple following the relation order. This is illustrated in Figure 5. Notice that when the second answer $(1, 1, 1, 3, 9)$ is popped, the current relation is $\text{rel}(4) = U$, with $R(1, 1)$, $S(1, 1)$, and $T(1, 3)$ from the earlier relations considered fixed (so we do not generate new answers from those relations). In fact, $(1, 1, 2)$ is already on the stack from the previous iteration. This logic ensures that we enumerate each query answer exactly once. For the time complexity, note that we visit each relation at most once in each iteration, thus the cost per iteration is $\tilde{O}(1)$ because query size is treated as a constant.

The LIFO nature of the stack is essential for achieving the lexicographic order. For instance, new answers that replace U -tuples (thus, change only the x_5 value) are always popped before answers that replace tuples in R , S , or T . In the following, we discuss the achievable orders in more detail.

3.3 Supported Lexicographic Orders

Different lexicographic orders can be achieved by different sortings of the individual relations. For example, if we sort R by $x_2 \rightarrow x_1$, we can achieve the order $x_2 \rightarrow x_1 \rightarrow x_3 \rightarrow x_4 \rightarrow x_5$ without any other change in the algorithm. Some other orders can be achieved

⁵Answers are represented as a tuple of values assigned to variables (x_1, \dots, x_5) , or alternatively, as a list of joining tuples $[t_1, \dots, t_4]$. For ease of presentation, we use the former in text and the latter in pseudocode.

⁶For an illustration, please see <https://www.youtube.com/watch?v=toi7ysuyRkw&t=1720s> [49].

Algorithm 2: Ranked enumeration for lexicographic orders without disruptive trios

Input: acyclic JQ Q , database D , lexicographic order L without disruptive trio

Output: Ranked enumeration of $Q(D)$ in L order

```

1 Remove self-joins from  $Q$  by copying the corresponding
  relations and renaming them in both  $D$  and  $Q$ 
2 Construct an  $L$ -consistent join tree  $\mathcal{T}$  of  $Q$  with
   $L$ -consistent relation order given by  $\text{rel}(i), i \in [\ell]$ 
3 Preprocess( $Q, D, \mathcal{T}, L, \text{rel}$ ) (Algorithm 1)
4 //A partial answer in the stack is represented as a list of
  input tuples together with their positions in the
  corresponding join groups (to easily get the next)
5 Initialize stack  $\mathcal{S}$  with element  $[(t_1, 1)]$  where  $t_1 = R[1]$  and
   $R$  is the root of  $\mathcal{T}$ 
6 repeat
7   //Pop a partial answer ( $1 \leq r \leq |\mathcal{T}|$ ), which also
   contains the positions  $j_i$  for each tuple  $t_i$ 
8    $s = \mathcal{S}.\text{pop}()$ ;  $[(t_1, j_1), \dots, (t_r, j_r)] = s$ 
9   //Look up matches in  $r$ -th relation
10   $M_r = \text{JoinIndex}_{\text{pr}(r) \rightarrow \text{rel}(r)}(t_{\text{pr}(r)})$ 
11  //Push partial answer with next tuple of  $r$ -th relation.
   It exists if  $j_r$  is not the last position in the group  $M_r$ 
12  if  $|M_r| \geq j_r + 1$  then
13     $s' = s.\text{copy}().\text{replaceLast}((M_r[j_r + 1], j_r + 1))$ 
14     $\mathcal{S}.\text{push}(s')$ 
15  //Range over the remaining relations
16  for  $i$  from  $r + 1$  to  $|\mathcal{T}|$  do
17    //Look up matches in  $i$ -th relation
18     $M_i = \text{JoinIndex}_{\text{pr}(i) \rightarrow \text{rel}(i)}(t_{\text{pr}(i)})$ 
19    //Extend partial answer with first tuple in matches
    of  $i$ -th relation
20     $t_i = M_i[1]$ ;  $s.\text{append}((t_i, 1))$ 
21    //s is now  $[(t_1, j_1), \dots, (t_r, j_r), \dots, (t_i, 1)]$ 
22    if  $|M_i| \geq 2$  then
23      //Push partial answer with next tuple of  $i$ -th
      relation to stack  $\mathcal{S}$ 
24       $s' = s.\text{copy}().\text{replaceLast}((M_i[2], 2))$ 
25       $\mathcal{S}.\text{push}(s')$ 
26  Merge  $s$  into single tuple and output
27 until query is interrupted or  $\mathcal{S}$  is empty

```

by additionally selecting a different topological sort on the join tree. With $[R, T, U, S]$ instead of $[R, S, T, U]$, we can achieve $x_1 \rightarrow x_2 \rightarrow x_4 \rightarrow x_5 \rightarrow x_3$. However, certain lexicographic orders cannot be achieved by this algorithm. Brault-Baron [15] identified a sufficient condition, which was later termed a *disruptive trio* [21] and shown to be necessary for other problems related to enumeration. (We discuss this in more detail in Section 5.4.)

DEFINITION 1 (DISRUPTIVE TRIO). For a CQ Q and lexicographic order L , three variables x_1, x_2, x_3 from L with relative order $x_1 \rightarrow x_2 \rightarrow x_3$ form a disruptive trio if x_1 and x_2 are not neighbors (i.e., they do not appear together in a Q atom), but x_3 is a neighbor of both x_1 and x_2 .

In our example, x_1, x_4, x_2 form a disruptive trio if L is $x_1 \rightarrow x_4 \rightarrow x_2$ or even $x_1 \rightarrow x_3 \rightarrow x_4 \rightarrow x_5 \rightarrow x_2$. Intuitively, during the enumeration, we cannot transition from R to T without fixing x_1, x_2 before x_4 ,

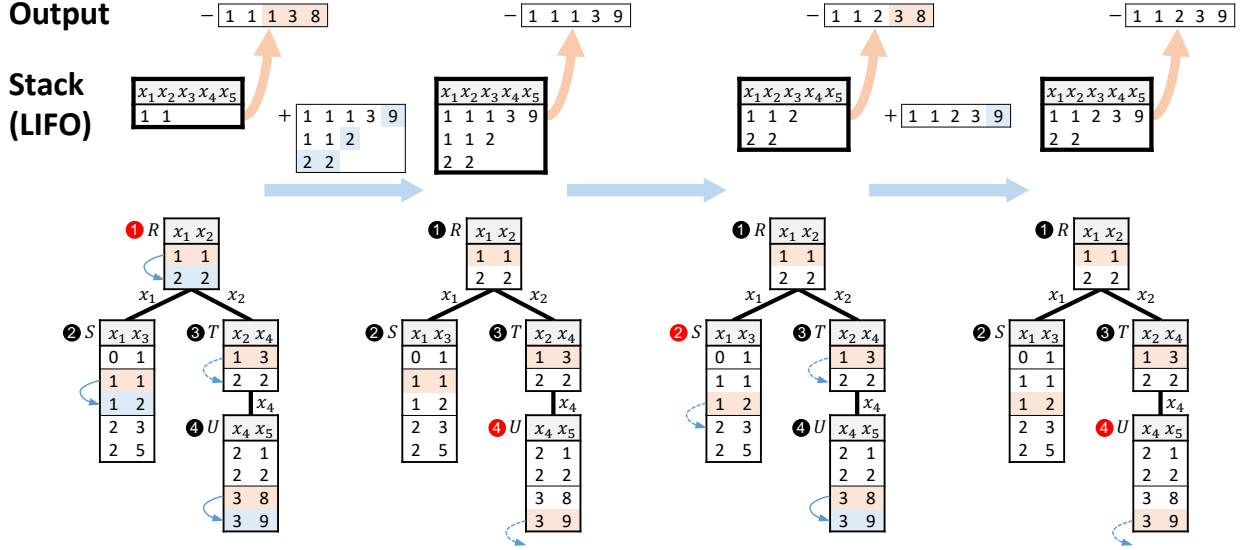


Figure 5: Enumeration steps for the first 4 answers by Algorithm 2 described in Section 3. The stack, shown on top, pops a partial answer, which is extended with the first matching tuples (in orange color) and moved to the output in each iteration. Starting from the last relation for which a tuple is in the partial answer (in red color), we check if a “next” tuple in the same join group exists (in blue color) and push a new answer to the stack. Dashed arrows indicate that there is no next.

which is inconsistent with the order L .

Braut-Baron showed that for lexicographic orders containing all free variables, the absence of a disruptive trio is equivalent to L being a *reverse alpha elimination order* [15, Theorem 15], and for partial lexicographic orders, it is equivalent to the lexicographic order being *consistent* with (or, in other words, a restriction of) a reverse alpha elimination order. An alpha elimination order is an ordering of the variables that guides the join tree construction [16]. If variable y follows variable x in the elimination order, then in the resulting join tree, y will never appear without x in any ancestor of a node that contains x .⁷ This guarantees that there exists a relation ordering \mathbf{rel} such that the L variables are encountered in the desired sequence. We call such an ordering of the relations, and its corresponding join tree, *L -consistent*.

If a desired lexicographic order has no disruptive trio, then we can find an L -consistent join tree and an L -consistent ordering of the relations to use with the enumeration algorithm discussed above.

THEOREM 2 (LEX). *Let Q be an acyclic join query over database D and L a lexicographic order of the variables in Q . If L does not contain a disruptive trio, then ranked enumeration of $Q(D)$ by*

⁷A similar property has been proposed in factorized databases in order to detect whether a lexicographic order is admissible with a given factorization order [7].

L can be achieved with $\text{TT}(k) = \tilde{O}(n + k)$.

Algorithm 2 shows the pseudocode. After the preprocessing phase, a loop returns query answers iteratively by popping and pushing from the stack. Notice that, for each answer, the algorithm keeps track of the positions j_1, \dots, j_ℓ of the tuples within the corresponding join group. This allows it to quickly access the next tuple in the group when constructing new answers (in Lines 13 and 24).

What about the lexicographic orders that contain disruptive trios? Algorithm 2 does not apply because there is no join tree that can match the order. For these orders, as well as SUM, we need a different strategy. Lexicographic orders with disruptive trios can in fact be reduced to a SUM-ordering problem by assigning the appropriate variable weights: If all relations have cardinality at most n , we can achieve that by setting the weight of the i^{th} value of the j^{th} variable in the order to $i \cdot n^{|L|-1-j}$.

4 Enumeration by SUM Order

In this section, we shift focus to ranking by SUM. Let $\sum_{i=1}^5 x_i$ (in ascending order) be the ranking function for our example query. A naive strategy is to select the best tuple from each relation based on its individual weight. For instance, using the same join tree as before, we could start with $R(1, 1)$ since it has the lowest weight $1 + 1 = 2$ within R . However, this strategy is not guaranteed to find the

top answers, at least not within the time bounds we aim for. Once we choose $R(1, 1)$, we will be stuck in a region of query answers with high overall weight because of the high weights of $U(3, 8)$ and $U(3, 9)$, which are the only matching tuples in U . The true top-1 answer $(2, 2, 3, 2, 1)$ starts with $R(2, 2)$, which matches with $U(2, 1)$. To make the right choices in R , the algorithm needs “lookahead” information about later matches in relations like U .

Unfortunately, it is infeasible to explicitly precompute the “lookahead” combinations of S, T, U tuples in the preprocessing phase, because that would exceed our desired $\tilde{O}(n)$. Instead, we rely on Dynamic Programming and a factorized representation of the query output. The enumeration phase is similar to the algorithm of Section 3.2, but uses a *priority queue* instead of a *stack* in order to prioritize the candidates according to the “lookahead” information computed during preprocessing.

4.1 Bottom-up Preprocessing Phase

To prioritize the tuples that lead to the lowest total weight, we modify the semijoin reduction so that, in addition to removing dangling tuples, we also compute the *best possible weight* $\text{opt}(t)$ reachable by each tuple t when joining it with other tuples in its subtree. This bottom-up computation is essentially a form of Dynamic Programming.

The algorithm is easier to present using tuple weights instead of attribute weights. We set the weights of R to $x_1 + x_2$, of S to x_3 , of T to x_4 and of U to x_5 . Such a conversion is always possible in linear time, which means that both regimes are supported in the algorithm. We only need to be careful so that the weight of each variable is assigned to a unique relation; this can be achieved through a mapping μ that assigns each variable x in the SUM to the first relation (or atom) that contains x in the topological sort rel . We denote the weight of tuple t by $\mathbf{w}(t)$. Algorithm 3 computes $\text{opt}(t)$ for all tuples t by aggregating the input weights using \min and $+$, bottom-up in reverse rel order $\textcircled{4} \rightarrow \textcircled{3} \rightarrow \textcircled{2} \rightarrow \textcircled{1}$, as shown in Figure 6. The leaf relations set $\text{opt}(t)$ to be equal to $\mathbf{w}(t)$. For tuple $T(2, 2)$, which is in the non-leaf relation T , we add its own weight 2 to the message $\min\{1, 2\}$ from the joining group in U , hence $\text{opt}(T(2, 2)) = 2 + \min\{1, 2\} = 3$. For a relation with multiple children, we add the messages from all of them. E.g., for $R(2, 2)$, we add its own weight 4 with the message $\min\{3, 5\}$ from S and the message $\min\{3\}$ from T , hence $\text{opt}(R(2, 2)) = 10$. By the end of the preprocessing step, we know the optimal weight $\text{opt}(t)$ for each tuple t , and the join index entries are *sorted* according to these values.

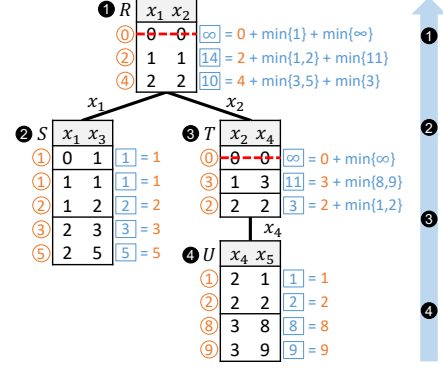


Figure 6: Bottom-up Dynamic Programming for SUM (preprocessing). Orange circles show the weights $\mathbf{w}(t)$ assigned to tuples. Blue squares show the calculated minimal subtree weight $\text{opt}(t)$ for each tuple.

Algorithm 3: Preprocessing for enumeration by SUM (Section 4.1). Changes compared to Algorithm 1 are in blue.

Input: acyclic JQ Q (without self-joins), database D with tuple weights, join tree \mathcal{T} , relation ordering rel consistent with \mathcal{T}
Output: reduced and sorted database D' , JoinIndex $_{R \rightarrow S}$ for each parent R and child S in \mathcal{T}
1 Initialize $\text{opt}(t) = \mathbf{w}(t)$ for all tuples t of all relations
2 $\text{rel} =$ relation ordering consistent with \mathcal{T}
3 **Process relations in reverse rel order (bottom-up in \mathcal{T})**
4 **for** $i = \ell$ down to 2 **do**
5 relation $S = \text{rel}[i]$; relation $R = \text{pr}(S)$
6 **Relation S has been reduced in a previous iteration (or is a leaf)**
7 Construct JoinIndex $_{R \rightarrow S}$ on shared attributes
8 Sort JoinIndex $_{R \rightarrow S}$ entries by opt
9 **for** tuple $t \in R$ **do**
10 $M = \text{JoinIndex}_{R \rightarrow S}(t)$
11 **Memoization: $\text{val}(M)$ is reused**
12 **if** $\text{opt}(M)$ not already computed **then**
13 $\text{opt}(M) = \min\{\infty, \min_{t' \in M} \text{opt}(t')\}$
14 $\text{opt}(t) = \text{opt}(t) + \text{opt}(M)$
15 **if** $\text{opt}(t) == \infty$ **then** remove t from R in D
16 Sort the root R by opt
17 **return** D , JoinIndex $_{R \rightarrow S}$ for all $(R, S) \in \mathcal{T}$

REMARK 1. The fact that Algorithm 3 is so similar to the semijoin reduction in Algorithm 1 is not a coincidence. They are both instances of the FAQ framework [2] with different semirings. In particular, the aggregation operators \vee and \wedge from the semi-join reduction are replaced with \min and $+$ in the variant for SUM. In more technical terms, the former corresponds to the Boolean semiring and the latter to the tropical semiring.

4.2 Top-down Enumeration Phase

As can be seen in Algorithm 4, the high-level logic of the enumeration is the same as the lexicographic enumeration of Algorithm 2. Lines 14 and 25 gen-

Algorithm 4: Ranked enumeration for SUM orders. Changes compared to Algorithm 2 are shown in blue.

Input: acyclic JQ Q , database D , SUM order \mathcal{W}
Output: Ranked enumeration of $Q(D)$ in \mathcal{W} order

- 1 Remove self-joins from Q by copying the corresponding relations and renaming them in both D and Q
- 2 Construct a join tree \mathcal{T} of Q with tree-consistent relation order given by $\text{rel}(i), i \in [\ell]$
- 3 Convert attribute weights to tuple weights
- 4 Preprocess($Q, D, \mathcal{T}, \text{rel}$) (Algorithm 3)
- 5 //A partial answer in the stack is represented as a list of input tuples together with their positions in the corresponding join groups (to easily get the next)
- 6 Initialize **priority queue** \mathcal{P} with element $[(t_1, 1)]$ where $t_1 = R[1]$ and R is the root of \mathcal{T}
- 7 **repeat**
- 8 //Pop a partial answer ($1 \leq r \leq |\mathcal{T}|$), which also contains the positions j_i for each tuple t_i
- 9 $s = \mathcal{S}.\text{pop}()$; $[(t_1, j_1), \dots, (t_r, j_r)] = s$
- 10 //Look up matches in r -th relation
- 11 $M_r = \text{JoinIndex}_{\text{pr}(r) \rightarrow \text{rel}(r)}(t_{\text{pr}(r)})$
- 12 //Push partial answer with next tuple of r -th relation. It exists if j_r is not the last position in the group M_r
- 13 **if** $|M_r| \geq j_r + 1$ **then**
- 14 $s' = s.\text{copy}().\text{replaceLast}((M_r[j_r + 1], j_r + 1))$
- 15 $\mathcal{P}.\text{push}(s')$ with **priority** $\text{prio}(s')$
- 16 //Range over the remaining relations
- 17 **for** i from $r + 1$ to $|\mathcal{T}|$ **do**
- 18 //Look up matches in i -th relation
- 19 $M_i = \text{JoinIndex}_{\text{pr}(i) \rightarrow \text{rel}(i)}(t_{\text{pr}(i)})$
- 20 //Extend partial answer with first tuple in matches of i -th relation
- 21 $t_i = M_i[1]$; $s.\text{append}((t_i, 1))$
- 22 // s is now $[(t_1, j_1), \dots, (t_r, j_r), \dots, (t_i, 1)]$
- 23 **if** $|M_i| \geq 2$ **then**
- 24 //Push partial answer with next tuple of i -th relation to **priority queue** \mathcal{P}
- 25 $s' = s.\text{copy}().\text{replaceLast}((M_i[2], 2))$
- 26 $\mathcal{P}.\text{push}(s')$ with **priority** $\text{prio}(s')$
- 27 Merge s into single tuple and output
- 28 **until** query is interrupted or \mathcal{P} is empty

erate new query answers with the tuple in the next position in the join group, like before. However, tuples within a join group are now sorted by opt , so each answer generated is guaranteed to produce the next-best weight (among those in the same join group) when extended to a complete answer.

Another important difference is that the stack that maintains the partial answers is replaced by a *priority queue* \mathcal{P} . Initially, \mathcal{P} only contains a partial answer with $R[1] = R(2, 2)$, producing the top-1 answer $(2, 2, 3, 2, 1)$ with weight $\text{opt}(R(2, 2)) = 10$. The second iteration has 3 candidates in \mathcal{P} : $(1, 1)$, $(2, 2, 5)$ and $(2, 2, 3, 2, 2)$. The priority of each candidate s , denoted by $\text{prio}(s)$ is the weight of the answer we will obtain if we fully extend it. We compute it before inserting it into \mathcal{P} ; we can either prematurely extend it into a full answer, or we can subtract from the previous answer the weight of the subtree that was removed and add the weight of the new subtree. For example, for $(2, 2, 5)$, we can

subtract the weight of the subtree rooted at $S(2, 3)$ and add the new weight $\text{opt}(S(2, 5))$ to the weight of the answer of the previous iteration, yielding $10 - 3 + 5 = 12$. Based on the priorities, $(2, 2, 3, 2, 2)$ with priority 11 will be the winner in the second iteration, and the enumeration continues accordingly. Figure 7 depicts the process.

The size of the priority queue \mathcal{P} is at most $k\ell$, since we push at most ℓ candidates in each iteration. Hence, the time of each iteration now includes a logarithmic cost for priority-queue operations (instead of the earlier constant one for stack accesses). However, if we ignore logarithmic factors, the $\text{TT}(k)$ complexity remains the same as in Theorem 2.

THEOREM 3 (SUM). *Let Q be an acyclic join query over database D and \mathcal{W} a SUM ranking function. Ranked enumeration of $Q(D)$ by \mathcal{W} can be achieved with $\text{TT}(k) = \tilde{O}(n + k)$.*

4.3 Performance in Practice

Any- k (enumeration by SUM) has been implemented and the experimental results from PVLDB'20 [44] and PVLDB'21 [47] have been independently reproduced.⁸ In Figure 8, we repeat and show an experiment from PVLDB'20 [44] that measures $\text{TT}(k)$ for a 4-path query (joining relations in a chain) on synthetic data.

The experiment compares ① Any- k against ② JOINFIRST (computing the full result with the Yannakakis algorithm [55]), and ③ PSQL (PostgreSQL 9.5.20). $\text{TT}(k)$ is depicted on the x-axis and k on the y-axis. By the time JOINFIRST returns the first answer (in 10.7 sec), any- k has already returned more than 4 million, starting with the first one after 67 msec. PSQL follows an approach similar to JOINFIRST and is outperformed for the top-ranked answers. For the last answer, any- k is slower by less than a factor of 3.

5 More General Queries and Tasks

We review a number of generalizations that have been studied, going beyond the task of ranked enumeration by SUM for acyclic JQs.

5.1 General Ranking Functions

Beyond lexicographic orders and SUM, the algorithm of Section 4 can be used with any ranking function that obeys a property called *subset-monotonicity*. Recall that a ranking function w maps the query answers to a domain W ordered by \preceq . We consider ranking functions that achieve

⁸The code is available to use at <https://github.com/northeastern-datalab/anyk-code>.

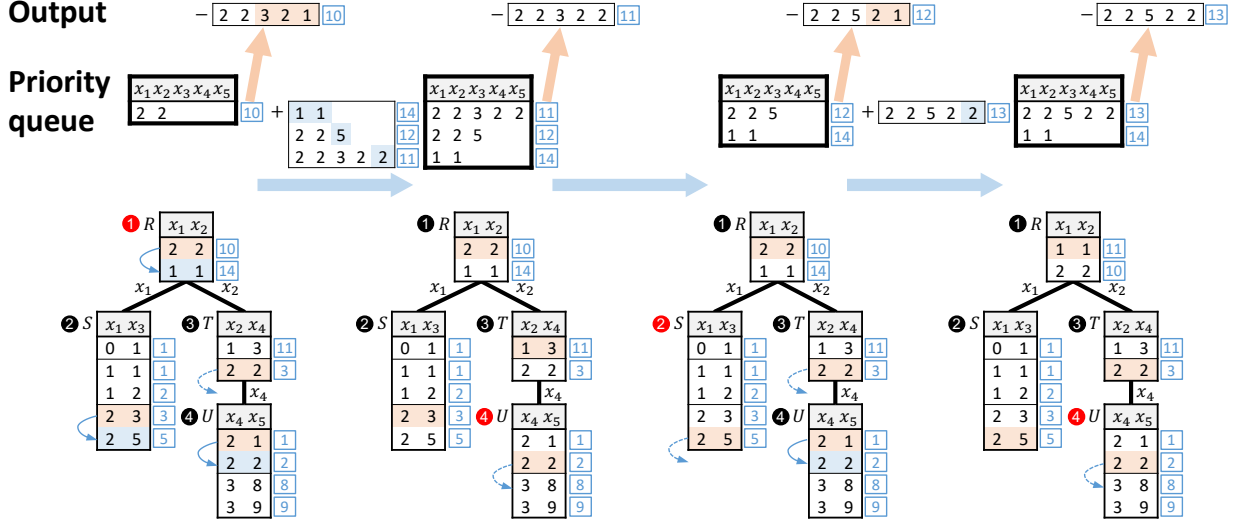


Figure 7: The example from Fig. 5 adapted with a priority queue instead of a stack to support ranked enumeration by SUM.

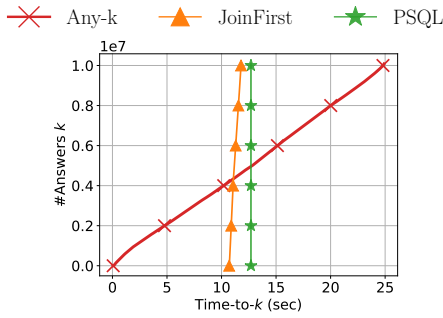


Figure 8: Any- k against the join-then-rank approach (JOINFIRST) and PostgreSQL (PSQL): Any- k returns the top answer in 67 msec, whereas JOINFIRST needs 10.7 sec [44].

this by aggregating (a multiset of) input weights via an aggregate function w_A . For example, w_A is \sum for SUM.

DEFINITION 4 (SUBSET-MONOTONICITY).
A ranking function w is subset-monotone if $w_A(X_1) \preceq w_A(X_2) \Rightarrow w_A(X_1 \uplus Y) \preceq w_A(X_2 \uplus Y)$ for all $X_1, X_2, Y \in \mathbb{N}^W$, where \uplus is multiset union.

Intuitively, subset-monotonicity allows to infer the ranking of complete solutions from the ranking of partial solutions. This is essentially enabling Dynamic Programming [10]. Any subset-monotone ranking function can be handled efficiently with the $\tilde{O}(n+k)$ guarantee for acyclic queries [43, 46].⁹ For

⁹Alternatively, the ranking function can be defined as a selective dioid [44], which can be shown to obey subset-monotonicity.

instance, we may choose the aggregate function to be max instead of \sum . Under this ranking, only the highest weight is relevant for ordering the answers.

Deep and Koutris [24] generalize subset-monotonicity¹⁰ so that the property is sensitive to the join tree structure; to achieve the desired guarantee, the property needs to hold only across the specific nodes of the join tree used by the algorithm. As an example, consider $f(x, y) + g(z)$ for arbitrary f, g and the query $Q(x, y, z) :- R(x, y), S(y, z)$. Even though this ranking function is not subset-monotone, it can be supported efficiently because x and y are encountered together.

What about other ranking functions? A known negative result is that if the ranking function is a black box, then one cannot do better than materializing the entire query output [24]. Thus, the only guarantee we can hope for is the worst-case output size of the query, given by the AGM bound [5].

5.2 CQs with Projection

So far, we have focused on join queries, yet CQs may also contain projection. Projections introduce a new challenge: even if ranked enumeration is efficient for a join query, this may not be true for projections, because we need to eliminate duplicates (under set semantics), potentially increasing $TT(k)$.

Bagan et al. [6] established a dichotomy for unranked enumeration that precisely characterizes queries that admit $TT(k) = \tilde{O}(n+k)$. The negative side of the dichotomy applies only to self-

¹⁰Subset-monotonicity is also referred to as a “totally decomposable ranking” [24].

join-free CQs and relies on two complexity-theoretic hypotheses: SPARSEBMM [11] states that two Boolean matrices A and B , represented as lists of non-zeros, cannot be multiplied in time $m^{1+o(1)}$ where m is the number of non-zeros in A , B , and AB . HYPERCLIQUE [1, 34] states that for every $k \geq 2$, there is no $O(n \text{ polylog } n)$ algorithm to decide the existence of a $(k+1, k)$ -hyperclique in a k -uniform hypergraph with n hyperedges, where a $(k+1, k)$ -hyperclique is a set of $k+1$ vertices such that every subset of k vertices forms a hyperedge, and a k -uniform hypergraph is one where all hyperedges contain exactly k vertices. Under these assumptions, the only efficient (self-join-free) CQs are those that are *free-connex*. A CQ is free-connex if it is acyclic and additionally, it remains acyclic if we add an atom that contains all free variables [15].

Interestingly, that frontier of tractability for unranked enumeration turns out to be the same for ranked enumeration with subset-monotone ranking functions (modulo logarithmic factors).

THEOREM 5 (DICHOTOMY [43, 46]). *Let Q be a CQ. If Q is free-connex, then ranked enumeration with a subset-monotone ranking function is possible with $\text{TT}(k) = \tilde{O}(n + k)$. Otherwise, if it is also self-join-free, then it is not possible with $\text{TT}(k) = \tilde{O}(n + k)$ for any ranking function, assuming SPARSEBMM and HYPERCLIQUE.*

For the class of acyclic but non-free-connex CQs, the dichotomy precludes the existence of an algorithm with the efficient $\tilde{O}(n + k)$ guarantee. However, $\tilde{O}(n \cdot k)$ is possible for subset-monotone ranking functions. This result has been established by the algorithm of Bagan et al. [6] for lexicographic orders, by Deep et al. [23] for lexicographic orders and SUM through a different algorithm, and by Kimelfeld and Sagiv [32] for all subset-monotone ranking functions through a third algorithm.

THEOREM 6 (NON-FREE-CONNEX [6, 23, 32]). *Let Q be an acyclic, non-free-connex CQ. Ranked enumeration of $Q(D)$ with a subset-monotone ranking function is possible with $\text{TT}(k) = \tilde{O}(n \cdot k)$.*

5.3 Beyond Acyclic CQs

We can apply the ranked-enumeration algorithms even to queries that are not acyclic CQs, albeit with adjusted complexity guarantees. This is possible if the query can be transformed into an acyclic and free-connex CQ, or a union of such queries. In that case, we first apply the transformation and then perform ranked enumeration on the resulting queries. To deal with a union, we maintain a top-level priority queue that retrieves the next query

answer from the query with the lowest weight in each iteration. Duplicate answers introduce potential complications, but as long as the number of duplicates per answer is bounded by a constant, they can be filtered on-the-fly without increasing complexity. In general, identifying such transformations is an orthogonal research problem, and we discuss three notable cases.

Cyclic JQs. For cyclic JQs, we can employ (hyper)tree decompositions [27] to reduce them to a union of acyclic JQs. A decomposition is associated with a width parameter that captures the degree of acyclicity of the query and affects the complexity of subsequent algorithms; for a JQ with width d , we can achieve $\text{TT}(k) = \tilde{O}(n^d + k)$. The state-of-the-art width for a JQ Q is the submodular width $\text{subw}(Q)$ [3, 35], transforming a cyclic JQ over a database of size n to a union of acyclic JQs of size $\tilde{O}(n^{\text{subw}(Q)})$, allowing ranked enumeration with $\text{TT}(k) = \tilde{O}(n^{\text{subw}(Q)} + k)$.¹¹

Built-in Predicates. Another case involves acyclic JQs that additionally contain built-in predicates [50] such as inequalities. For *non-equalities* (or “disequalities” \neq), we can always achieve $\text{TT}(k) = \tilde{O}(n + k)$ regardless of where the non-equalities appear in the JQ through a “color-coding” technique [39]. Abo Khamis et al. [31] showed that the same is true for a multidimensional generalization of non-equality, called a Not-All-Equal (NAE) predicate. For *inequalities* ($<$, $>$), we can successfully reduce the query to an acyclic JQ over an $\tilde{O}(n)$ database, hence achieving $\text{TT}(k) = \tilde{O}(n + k)$, as long as the inequality predicate involves variables that appear in join-tree nodes that are adjacent [47]. This condition can be checked directly from the query structure; it is equivalent to the absence of a chordless path of length at least 4 connecting the inequality variables, in the query’s hypergraph [45].

CQs with FDs. While a CQ may be acyclic but not free-connex, or even cyclic, it may still be possible to transform it to an acyclic CQ without employing a hypertree decomposition, which generally increase the complexity. This is the case when Functional Dependencies (FDs) are present in the CQ. We can achieve $\text{TT}(k) = \tilde{O}(n + k)$ for queries whose so-called *FD-extension*, also known as the *closure* of Q [26], is free-connex [19].

5.4 Direct Access

A problem that is closely related to ranked enumeration is *direct access* [21, 22, 25], which asks whether it is possible to efficiently jump to arbitrary

¹¹An analog exists for CQs (with projection) [12].

positions in the (implicit) output array, after a preprocessing phase. Ranked enumeration is a special case of this problem, where the accessed positions are $1, 2, 3, \dots$

Interestingly, the absence of disruptive trios (Definition 1) that describes the feasible lexicographic orders for the algorithm of Section 3 also appears as a necessary condition for achieving direct access with quasilinear preprocessing and polylogarithmic delay [21] (assuming SPARSEBMM). The other necessary condition is for the (self-join-free) acyclic CQ to be L -connex for the variables L that appear in the lexicographic order; similarly to the free-connex property, this means that the query remains acyclic when we add a hyperedge consisting of the L variables. These two conditions are also sufficient for acyclic CQs and thus, provide a dichotomy for self-join-free CQs, under SPARSEBMM and HYPERCLIQUE.

A similar, but much more restrictive on the positive side, dichotomy has been established for SUM [21]. Going beyond quasilinear preprocessing time, Bringman et al. [17] derived precise bounds for each JQ and lexicographic order, Eldar et al. [25] considered queries with aggregation, while Tziavelis et al. [45] studied the problem of a single access where no preprocessing is required.

6 Conclusion and Future Outlook

In this paper, we explored the problem of ranked enumeration without fully materializing the query result. We discussed how, for acyclic join queries, certain lexicographic orders can naturally be produced by the unranked enumeration algorithm (with an additional sorting of individual relations). However, not all orders can be handled in this straightforward way. With additional preprocessing and data structures for prioritization, we presented an extended algorithm capable of handling more complex ranking functions, including SUM. Notably, for free-connex CQs, this approach achieves $TT(k) = \tilde{O}(n+k)$ for any subset-monotone ranking function, and no other self-join-free CQ admits this guarantee (under common hypotheses). Broader classes of queries are also within the reach of the algorithm, as long as they can be efficiently reduced to a union of acyclic and free-connex CQs.

These results are part of an extensive line of research in database theory, focused on the computational tasks that can be efficiently performed on query results without explicitly materializing them. The goal is to offer the illusion of a materialized result, while the actual operations are executed directly on the database. Beyond ranked enumeration

and direct access, related tasks include aggregation [2], linear regression [37], and k -means clustering [36], among others.

One of the areas lacking a refined understanding for ranked enumeration is the complexity landscape for ranking functions. Although some orders are algorithmically easier to achieve than others within the subset-monotone class, their complexity is the same, modulo logarithmic factors. On the other end of the spectrum, for arbitrary black-box ranking functions, no strong guarantees can be achieved. What about the space in-between? To contrast this with the problem of direct access, more intriguing, polynomial-time separations are known even within the class of lexicographic and SUM ranking functions. Mapping out properties of ranking functions and their impact on complexity is an interesting research direction.

Similarly, more work is needed to understand the fundamental difficulty of ranking. For instance, are there surprising cases where ranked enumeration is harder than unranked? One avenue to approach this question is to study CQs with “long” inequalities (in contrast to the “short” inequalities of Section 5.3). For queries, such as $Q(x_1, x_2, x_3, x_4) :- R(x_1, x_2), S(x_2, x_3), T(x_3, x_4), x_1 < x_4$, it is known that unranked enumeration can be achieved with $\tilde{O}(n+k)$ [52], yet ranked enumeration has not been studied. Another avenue is to consider different classes of circuits [4] instead of CQs in order to find such a separation.

The relationship between ranked enumeration and top- k can also lead to interesting questions. Top- k introduces two relaxations, the exact impact of which is not entirely clear: (1) k is a small constant, and (2) k is known in advance.

Finally, parallelization is a natural, but challenging, direction. The prioritization of answers required by ranked enumeration implies a degree of sequentiality in the computation, making a parallel adaptation non-obvious. On the theoretical side, the widely used MPC model [8] does not seem to be a good fit because of its batch-processing nature.

Acknowledgements. This work was supported in part by a grant from PricewaterhouseCoopers (PwC), the National Institutes of Health (NIH) under award number R01 NS091421, and by the National Science Foundation (NSF) under award numbers CAREER IIS-1762268 and IIS-1956096. Nikolaos Tziavelis was additionally supported by a Google PhD fellowship. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

- [1] A. Abboud and V. V. Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *FOCS*, pages 434–443, 2014. DOI: [10.1109/FOCS.2014.53](https://doi.org/10.1109/FOCS.2014.53).
- [2] M. Abo Khamis, H. Q. Ngo, and A. Rudra. FAQ: questions asked frequently. In *PODS*, pages 13–28, 2016. DOI: [10.1145/2902251.2902280](https://doi.org/10.1145/2902251.2902280).
- [3] M. Abo Khamis, H. Q. Ngo, and D. Suciu. What do shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another? In *PODS*, pages 429–444, 2017. DOI: [10.1145/3034786.3056105](https://doi.org/10.1145/3034786.3056105).
- [4] A. Amarilli, P. Bourhis, F. Capelli, and M. Monet. Ranked Enumeration for MSO on Trees via Knowledge Compilation. In *ICDT*, volume 290, 25:1–25:18, 2024. DOI: [10.4230/LIPIcs.ICDT.2024.25](https://doi.org/10.4230/LIPIcs.ICDT.2024.25).
- [5] A. Atserias, M. Grohe, and D. Marx. Size bounds and query plans for relational joins. *SIAM J. Comput.*, 42(4):1737–1767, 2013. DOI: [10.1137/110859440](https://doi.org/10.1137/110859440).
- [6] G. Bagan, A. Durand, and E. Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *International Workshop on Computer Science Logic (CSL)*, pages 208–222, 2007. DOI: [10.1007/978-3-540-74915-8_18](https://doi.org/10.1007/978-3-540-74915-8_18).
- [7] N. Bakibayev, T. Kočíský, D. Olteanu, and J. Závodný. Aggregation and ordering in factorised databases. *PVLDB*, 6(14):1990–2001, 2013. DOI: [10.14778/2556549.2556579](https://doi.org/10.14778/2556549.2556579).
- [8] P. Beame, P. Koutris, and D. Suciu. Communication steps for parallel query processing. *J. ACM*, 64(6), 2017. DOI: [10.1145/3125644](https://doi.org/10.1145/3125644).
- [9] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30(3):479–513, 1983. DOI: [10.1145/2402.322389](https://doi.org/10.1145/2402.322389).
- [10] R. Bellman. The theory of dynamic programming. *Bull. Amer. Math. Soc.*, 60(6):503–515, Nov. 1954. URL: <https://projecteuclid.org/443/euclid.bams/1183519147>.
- [11] C. Berkholz, F. Gerhardt, and N. Schweikardt. Constant delay enumeration for conjunctive queries: a tutorial. *ACM SIGLOG News*, 7(1):4–33, 2020. DOI: [10.1145/3385634.3385636](https://doi.org/10.1145/3385634.3385636).
- [12] C. Berkholz and N. Schweikardt. Constant delay enumeration with FPT-preprocessing for conjunctive queries of bounded submodular width. In *MFCS*, volume 138 of *LIPIcs*, 58:1–58:15, 2019. DOI: [10.4230/LIPIcs.MFCS.2019.58](https://doi.org/10.4230/LIPIcs.MFCS.2019.58).
- [13] P. A. Bernstein and D. W. Chiu. Using semi-joins to solve relational queries. *J. ACM*, 28(1):25–40, 1981. DOI: [10.1145/322234.322238](https://doi.org/10.1145/322234.322238).
- [14] M. S. Boddy. Anytime problem solving using dynamic programming. In *AAAI*, pages 738–743, 1991. URL: <https://dl.acm.org/doi/abs/10.5555/1865756.1865791>.
- [15] J. Brault-Baron. *De la pertinence de l'énumération: complexité en logiques propositionnelle et du premier ordre*. PhD thesis, Université de Caen, 2013. URL: <https://hal.archives-ouvertes.fr/tel-01081392>.
- [16] J. Brault-Baron. Hypergraph acyclicity revisited. *ACM Comput. Surv.*, 49(3), Dec. 2016. DOI: [10.1145/2983573](https://doi.org/10.1145/2983573).
- [17] K. Bringmann, N. Carmeli, and S. Mengel. Tight fine-grained bounds for direct access on join queries. In *PODS*, 427–436, 2022. DOI: [10.1145/3517804.3526234](https://doi.org/10.1145/3517804.3526234).
- [18] F. Capelli and Y. Strozecki. Geometric Amortization of Enumeration Algorithms. In *STACS 2023*, volume 254, 18:1–18:22, 2023. DOI: [10.4230/LIPIcs.STACS.2023.18](https://doi.org/10.4230/LIPIcs.STACS.2023.18).
- [19] N. Carmeli and M. Kröll. Enumeration complexity of conjunctive queries with functional dependencies. *Theory Comput. Syst.*, 64(5):828–860, 2020. DOI: [10.1007/s00224-019-09937-9](https://doi.org/10.1007/s00224-019-09937-9).
- [20] N. Carmeli and M. Kröll. On the enumeration complexity of unions of conjunctive queries. *TODS*, 46(2), 2021. DOI: [10.1145/3450263](https://doi.org/10.1145/3450263).
- [21] N. Carmeli, N. Tziavelis, W. Gatterbauer, B. Kimelfeld, and M. Riedewald. Tractable orders for direct access to ranked answers of conjunctive queries. *TODS*, 48(1), 2023. DOI: [10.1145/3578517](https://doi.org/10.1145/3578517).
- [22] 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*, pages 393–409, 2020. DOI: [10.1145/3375395.3387662](https://doi.org/10.1145/3375395.3387662).
- [23] S. Deep, X. Hu, and P. Koutris. Ranked enumeration of join queries with projections. *PVLDB*, 15(5):1024–1037, 2022. DOI: [10.14778/3510397.3510401](https://doi.org/10.14778/3510397.3510401).
- [24] S. Deep and P. Koutris. Ranked enumeration of conjunctive query results. In *ICDT*, volume 186, 5:1–5:19, 2021. DOI: [10.4230/LIPIcs.ICDT.2021.5](https://doi.org/10.4230/LIPIcs.ICDT.2021.5).
- [25] I. Eldar, N. Carmeli, and B. Kimelfeld. Direct Access for Answers to Conjunctive Queries with Aggregation. In *ICDT*, volume 290, 4:1–4:20, 2024. DOI: [10.4230/LIPIcs.ICDT.2024.4](https://doi.org/10.4230/LIPIcs.ICDT.2024.4).
- [26] W. Gatterbauer and D. Suciu. Dissociation and propagation for approximate lifted inference with standard relational database management systems. *The VLDB Journal*, 26(1):5–30, 2017. DOI: [10.1007/s00778-016-0434-5](https://doi.org/10.1007/s00778-016-0434-5).
- [27] G. Gottlob, G. Greco, N. Leone, and F. Scarcello. Hypertree decompositions: questions and answers. In *PODS*, pages 57–74, 2016. DOI: [10.1145/2902251.2902309](https://doi.org/10.1145/2902251.2902309).
- [28] M. Idris, M. Ugarte, S. Vansummeren, H. Voigt, and W. Lehner. General dynamic Yannakakis: conjunctive queries with theta joins under updates. *VLDB J.*, 29:619–653, 2020. DOI: [10.1007/s00778-019-00590-9](https://doi.org/10.1007/s00778-019-00590-9).

- [29] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top- k query processing techniques in relational database systems. *ACM Computing Surveys*, 40(4):11, 2008. DOI: [10.1145/1391729.1391730](https://doi.org/10.1145/1391729.1391730).
- [30] 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)*, pages 15–29. Springer, 1999. DOI: [10.1007/3-540-48318-7_4](https://doi.org/10.1007/3-540-48318-7_4).
- [31] M. A. Khamis, H. Q. Ngo, D. Olteanu, and D. Suciu. Boolean tensor decomposition for conjunctive queries with negation. In *ICDT*, 21:1–21:19, 2019. DOI: [10.4230/LIPIcs.ICDT.2019.21](https://doi.org/10.4230/LIPIcs.ICDT.2019.21).
- [32] B. Kimelfeld and Y. Sagiv. Incrementally computing ordered answers of acyclic conjunctive queries. In *International Workshop on Next Generation Information Technologies and Systems (NGITS)*, pages 141–152, 2006. DOI: [10.1007/11780991_13](https://doi.org/10.1007/11780991_13).
- [33] 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*, 18(7):401–405, 1972. DOI: [10.1287/mnsc.18.7.401](https://doi.org/10.1287/mnsc.18.7.401).
- [34] A. Lincoln, V. V. Williams, and R. R. Williams. Tight hardness for shortest cycles and paths in sparse graphs. In *SODA*, pages 1236–1252, 2018. DOI: [10.1137/1.9781611975031.80](https://doi.org/10.1137/1.9781611975031.80).
- [35] D. Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *J. ACM*, 60(6):42:1–42:51, 2013. DOI: [10.1145/2535926](https://doi.org/10.1145/2535926).
- [36] B. Moseley, K. Pruhs, A. Samadian, and Y. Wang. Relational Algorithms for k -Means Clustering. In *ICALP*, volume 198, 97:1–97:21, 2021. DOI: [10.4230/LIPIcs.ICALP.2021.97](https://doi.org/10.4230/LIPIcs.ICALP.2021.97).
- [37] D. Olteanu and M. Schleich. Factorized databases. *SIGMOD Record*, 45(2), 2016. DOI: [10.1145/3003665.3003667](https://doi.org/10.1145/3003665.3003667).
- [38] D. Olteanu and J. Závodný. Size bounds for factorised representations of query results. *TODS*, 40(1):2, 2015. DOI: [10.1145/2656335](https://doi.org/10.1145/2656335).
- [39] C. H. Papadimitriou and M. Yannakakis. On the complexity of database queries. *Journal of Computer and System Sciences*, 58(3):407–427, 1999. DOI: [10.1006/jcss.1999.1626](https://doi.org/10.1006/jcss.1999.1626).
- [40] R. Paredes and G. Navarro. *Optimal incremental sorting*. In *ALENEX*. 2006, pages 171–182. DOI: [10.1137/1.9781611972863.16](https://doi.org/10.1137/1.9781611972863.16).
- [41] L. Segoufin. Constant delay enumeration for conjunctive queries. *SIGMOD Record*, 44(1):10–17, 2015. DOI: [10.1145/2783888.2783894](https://doi.org/10.1145/2783888.2783894).
- [42] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13(3):566–579, 1984. DOI: [10.1137/0213035](https://doi.org/10.1137/0213035).
- [43] N. Tziavelis. *Efficient Ranked Access to Database Query Answers*. PhD thesis, Northeastern University, 2024. URL: <https://hdl.handle.net/2047/D20668633>.
- [44] N. Tziavelis, D. Ajwani, W. Gatterbauer, M. Riedewald, and X. Yang. Optimal algorithms for ranked enumeration of answers to full conjunctive queries. *PVLDB*, 13(9):1582–1597, 2020. DOI: [10.14778/3397230.3397250](https://doi.org/10.14778/3397230.3397250).
- [45] N. Tziavelis, N. Carmeli, W. Gatterbauer, B. Kimelfeld, and M. Riedewald. Efficient computation of quantiles over joins. In *PODS*, pages 303–315, 2023. DOI: [10.1145/3584372.3588670](https://doi.org/10.1145/3584372.3588670).
- [46] N. Tziavelis, W. Gatterbauer, and M. Riedewald. Any- k algorithms for enumerating ranked answers to conjunctive queries. *CoRR*, abs/2205.05649, 2023. URL: <https://arxiv.org/abs/2205.05649>.
- [47] N. Tziavelis, W. Gatterbauer, and M. Riedewald. Beyond equi-joins: ranking, enumeration and factorization. *PVLDB*, 14(11):2599–2612, 2021. DOI: [10.14778/3476249.3476306](https://doi.org/10.14778/3476249.3476306).
- [48] N. Tziavelis, W. Gatterbauer, and M. Riedewald. Optimal join algorithms meet top- k . In *SIGMOD tutorials*, pages 2659–2665, 2020. DOI: [10.1145/3318464.3383132](https://doi.org/10.1145/3318464.3383132). URL: <https://northeastern-datalab.github.io/topk-join-tutorial/>.
- [49] N. Tziavelis, W. Gatterbauer, and M. Riedewald. Toward responsive DBMS: optimal join algorithms, enumeration, factorization, ranking, and dynamic programming. In *ICDE tutorials*, 2022. DOI: [10.1109/ICDE53745.2022.00299](https://doi.org/10.1109/ICDE53745.2022.00299). URL: <https://northeastern-datalab.github.io/responsive-dbms-tutorial/>.
- [50] J. D. Ullman. *Principles of database and knowledge-base systems, Vol. I*. Computer Science Press, Inc., 1988. URL: <https://dl.acm.org/doi/abs/10.5555/42790>.
- [51] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC*, pages 137–146, 1982. DOI: [10.1145/800070.802186](https://doi.org/10.1145/800070.802186).
- [52] Q. Wang and K. Yi. Conjunctive queries with comparisons. In *SIGMOD*, pages 108–121, 2022. DOI: [10.1145/3514221.3517830](https://doi.org/10.1145/3514221.3517830).
- [53] 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*, pages 489–498, 2018. DOI: [10.1145/3178876.3186115](https://doi.org/10.1145/3178876.3186115).
- [54] 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)*, pages 1–3, 2018. DOI: [10.1145/3214708.3214711](https://doi.org/10.1145/3214708.3214711).
- [55] M. Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, pages 82–94, 1981. URL: <https://dl.acm.org/doi/10.5555/1286831.1286840>.

Budget-aware Query Tuning: An AutoML Perspective

Wentao Wu Chi Wang

Microsoft Research

{wentao.wu, wang.chi}@microsoft.com

ABSTRACT

Modern database systems rely on cost-based query optimizers to come up with good execution plans for input queries. Such query optimizers rely on *cost models* to estimate the costs of candidate query execution plans. A cost model represents a function from a set of *cost units* to query execution cost, where each cost unit specifies the *unit cost* of executing a certain type of query processing operation (such as table scan or join). These cost units are traditionally viewed as *constants*, whose values only depend on the platform configuration where the database system runs on top of but are invariant for queries processed by the database system. In this paper, we challenge this classic view by thinking of these cost units as *variables* instead. We show that, by varying the cost-unit values one can obtain query plans that significantly outperform the default query plans returned by the query optimizer when viewing the cost units as constants. We term this cost-unit tuning process “query tuning” (QT) and show that it is similar to the well-known hyper-parameter optimization (HPO) problem in AutoML. As a result, any state-of-the-art HPO technologies can be applied to QT. We study the QT problem in the context of *anytime tuning*, which is desirable in practice by constraining the total time spent on QT within a given budget—we call this problem *budget-aware* query tuning. We further extend our study from tuning a single query to tuning a workload with multiple queries, and we call this generalized problem budget-aware workload tuning (WT), which aims for minimizing the execution time of the entire workload. WT is more challenging as one needs to further prioritize individual query tuning within the given time budget. We propose solutions to both QT and WT and experimental evaluation using both benchmark and real workloads demonstrates the efficacy of our proposed solutions.

1. INTRODUCTION

Modern database systems rely on cost-based query optimizers to come up with good execution plans for input queries. Such query optimizers rely on *cost models*

Cost Unit	Default Value
<i>seq_page_cost</i>	1.0
<i>random_page_cost</i>	4.0
<i>cpu_tuple_cost</i>	0.01
<i>cpu_index_tuple_cost</i>	0.005
<i>cpu_operator_cost</i>	0.0025
<i>parallel_tuple_cost</i>	0.1

Table 1: Examples of cost units used by PostgreSQL’s query planner/optimizer [1].

to estimate costs of candidate query execution plans. A cost model represents a function from a set of *cost units* to the query execution cost, where each cost unit specifies the *unit cost* of executing a certain type of query processing operation (such as table scan or join). For instance, Table 1 presents some of the cost units used by PostgreSQL’s query optimizer [1]. This paradigm is followed by other mainstream query processing engines as well, such as Microsoft SQL Server [8], IBM DB2 [25], and Apache Spark [15]. These cost units are traditionally viewed as *constants* [1], whose values only depend on the *platform configuration* (e.g., CPU speed) where the query processing engine runs on top of and need to be *calibrated* against the platform [11, 26, 34, 35, 36, 38]; however, they are *invariant* regardless of the queries being processed by the database system.

In this paper, we challenge this classic view by thinking of these cost units as *variables* instead that can be changed across queries. We show that, by varying the cost-unit values one can obtain query plans that significantly outperform the default query plans returned by the query optimizer when viewing the cost units as constants. We call this per-query cost-unit tuning process “query tuning” (QT). At a high-level, query tuning is similar to the *hyper-parameter optimization* (HPO) problem [39] from *automated machine learning* (a.k.a. AutoML), which has received tremendous attention in recent years [13]. While the HPO problem aims to find appropriate hyper-parameter values, such as the learning rate and batch size when using stochastic gradient descent (SGD) to train a deep neural network, that can improve the quality of the ML model, QT shares the similar goal of improving the quality of the query plan by seeking appropriate values of the cost units.

The similarity between QT and HPO implies that existing HPO technologies can be directly applied to QT. However, a straightforward application is less attractive, as most of the existing HPO technologies are not *budget-aware*, namely, they do not constrain themselves to conform to a user-specified *tuning time budget*. There are so far only a few exceptions, including Hyperband [20], BOHB [12], CFO [33], and BlendSearch [30], which allow user to specify a *timeout* and will exit HPO once the timeout is reached. However, in practice one often wants to tune multiple queries (called a workload) altogether instead of tuning just one single query. None of the above work on HPO can be directly applied to this multi-query workload tuning (WT) problem.

In this paper, we study the budget-aware QT and WT problems. We propose solutions to both problems and evaluate their efficacy using both benchmark and real workloads. The query plans found by our current solutions can significantly improve over the default query plans generated by the query optimizer while conforming to the tuning time budget.

The idea of tuning query performance has been extensively explored in the literature but in different sense compared to the one proposed in this paper. Lots of recent work has been devoted to the so-called *knob tuning* (see [40] for a recent survey). Some existing work, e.g., UDO [31], also views the cost units studied in this paper as tunable knobs, though we think the cost units should be separated from other knobs and are worth its own treatment for the following reasons. First, some of the knobs are related to the runtime configurations of the database server, e.g., buffer pool size. Changing those knobs often requires a server restart that may not be feasible in many situations (e.g., cloud database services with stringent SLA's). In contrast, tuning the cost units does not require a server restart and therefore does not pose significant impact on the runtime state of the database server. Second, while there are also other knobs that do not require server restart, e.g., *max_parallel_workers* for PostgreSQL, such knobs only affect the running time of the query plan that has already been chosen by the query optimizer. In contrast, cost units can affect the decision made by the query optimizer in terms of choosing which query plan for execution. This is indeed a more fundamental distinction when viewing cost units as tunable knobs. In this spirit, the cost units can be thought of as a certain type of query hint [7]. However, the goal of query hint is to constrain the search space of the query optimizer to avoid bad query plans that could have been proposed by the optimizer without such restrictions; on the other hand, tuning cost units actually gives the optimizer more freedom in terms of proposing query plans, and the decision of picking the best plan is deferred to the mo-

ment when actual plan execution time is observed. Due to their overheads, the query/workload tuning technologies studied in this paper can perhaps only be applied to *recurring* queries/workloads, where one can tune the queries/workloads in an offline manner and pay it as a one-time price [14]. Nonetheless, given the strong connection between QT and HPO, we believe that more progress can be made in the future to enable QT for tuning more adhoc workloads in an online fashion.

2. PROBLEM FORMULATION

We assume that a query optimizer uses a cost model configured with a set of tunable parameters, referred to as *cost units* (ref. Table 1 for examples), to estimate the cost of a candidate query execution plan. The plan with the lowest estimated cost is returned by the optimizer for final query execution. Without loss of generality, we use \vec{u} to represent the set of tunable cost units as an ordered vector. Given a query q , we use $P(q, \vec{u})$ to represent the query plan returned by the query optimizer with the cost units \vec{u} . Different values of \vec{u} may therefore yield different query plans. We next formulate the problems of budget-aware query tuning and workload tuning.

2.1 Budget-aware Query Tuning

Let q be a specific query, and let U be the search space of \vec{u} . That is, if $\vec{u} = (u_1, \dots, u_m)$ where each u_i is within some range/domain D_i (e.g., $u_i \in [a_i, b_i]$), then $U = \prod_{i=1}^m D_i$. This covers both discrete and continuous cost units, though in practice cost units are typically continuous (within certain ranges).

Let B be a given budget on the tuning time. Let $\vec{u}_1, \dots, \vec{u}_K$ be the successive trials on the cost units. Let $t(q, \vec{u}_j)$ be the execution time of the corresponding query plan $P(q, \vec{u}_j)$, for $1 \leq j \leq K$. The *budget constraint* can then be expressed as $\sum_{j=1}^K t(q, \vec{u}_j) \leq B$. The problem of budget-aware query tuning is defined as:

Definition 1 (Budget-aware Query Tuning). Find $\vec{u}^* = \operatorname{argmin}_{1 \leq j \leq K} \{t(q, \vec{u}_j)\}$ w.r.t. $\sum_{j=1}^K t(q, \vec{u}_j) \leq B$.

2.2 Budget-aware Workload Tuning

Let $W = \{q_1, \dots, q_n\}$ be a workload of n queries. Let f_i be the *frequency* of the query q_i being tuned. For q_i , we define its total tuning time $t_i = \sum_{j=1}^{f_i} t(q_i, \vec{u}_{ij})$, where \vec{u}_{ij} represents (the cost units of) the j -th trial of q_i . The budget constraint at workload-level can then be expressed as $\sum_{i=1}^n t_i \leq B$.

Definition 2 (Budget-aware Workload Tuning). Find

$$\vec{u}_i^* = \operatorname{argmin}_{1 \leq j \leq f_i} \{t(q_i, \vec{u}_{ij})\}$$

for $1 \leq i \leq n$ w.r.t. the budget constraint $\sum_{i=1}^n t_i \leq B$.

Remark. The above definition automatically minimizes the workload execution time w.r.t. the budget constraint

on tuning time, which can be expressed as

$$t(W, \vec{u}_W^*) = t(q_1, \dots, q_n, \vec{u}_1^*, \dots, \vec{u}_n^*) = \sum_{i=1}^n t(q_i, \vec{u}_i^*).$$

Discussion. An alternative of Definition 2 is to find one set of cost units for the entire workload, instead of one set of cost units for each individual query. Specifically, let $\vec{u}_1, \dots, \vec{u}_K$ be the successive trials on the cost units for the entire workload W , and let $t(W, \vec{u}_j) = \sum_{i=1}^n t(q_i, \vec{u}_j)$ be the total workload execution time.

Definition 3 (Query-independent Budget-aware Workload Tuning). Find $\vec{u}^* = \operatorname{argmin}_{1 \leq j \leq K} \{t(W, \vec{u}_j)\}$ w.r.t. the budget constraint $\sum_{j=1}^K t(W, \vec{u}_j) \leq B$.

When the budget B is sufficient, Definition 2 is more general than Definition 3, because we should be able to get the same or a better query plan for each query under Definition 2. To see this, notice that the best sets of cost units for individual queries are not necessarily the same, which, on the other hand, is an implicit constraint under Definition 3. It remains interesting to empirically compare Definitions 2 and 3 when budget is limited, and we leave this as one direction for future work.

3. PROPOSED SOLUTIONS

Below we propose solutions to budget-aware query tuning (QT) and workload tuning (WT).

3.1 Budget-aware Query Tuning

Since QT is similar to HPO, in theory any HPO algorithm \mathcal{A} can be adapted to work for QT. For example, *random search* is a simple but competitive algorithm for HPO [5]. It can be easily customized to a budget-aware algorithm by monitoring the time spent on each random trial and terminating once the timeout is reached.

Optimization by Plan Caching. Since different cost units may result in the same query execution plan, it is possible that some plan $P(q, \vec{u}_j)$ is a duplicate of another plan $P(q, \vec{u}_i)$ ($i < j$) in the sequence $\vec{u}_1, \dots, \vec{u}_K$. One optimization is therefore to maintain a *cache* for observed plans, if memory is not constrained. The tuning time of a duplicate plan is set to zero.

Optimization by Early Stopping. Since only the optimal \vec{u}^* matters, we can safely stop executing a plan $P(q, \vec{u}_j)$ if $t(q, \vec{u}_j) \geq t(q, \vec{u}_j^*)$, where $t(q, \vec{u}_j^*)$ is the lowest execution time observed up to the j -th trial. In practice, we usually have a default value \vec{u}_0 for \vec{u} (e.g., the built-in values such as the ones shown in Table 1 for PostgreSQL). As a special case of the above “early stopping” idea, we can stop executing $P(q, \vec{u}_j)$ if $t(q, \vec{u}_j) \geq t(q, \vec{u}_0)$. This is also a worst-case scenario, as we have $t(q, \vec{u}_j^*) \leq t(q, \vec{u}_0)$, obviously.

3.2 Budget-aware Workload Tuning

WT is more complicated than QT, as one has to decide which query to tune next while conforming to the

total budget on tuning time. We propose the following four strategies: (1) round robin; (2) cost-based prioritization; (3) multi-armed bandit; (4) improvement rate. Moreover, both the plan-cache based optimization and the early-stopping optimization can be used for WT.

3.2.1 Round Robin

The round robin strategy simply rotates among the queries and stops when the tuning time budget is exhausted. Albeit a simple strategy, it has been deemed as a robust and strong baseline in the literature [21].

3.2.2 Cost-based Prioritization

This strategy is inspired by the idea of using a priority queue to prioritize query tuning in Microsoft’s Database Tuning Advisor (DTA) [9]. Specifically, we order the queries by their best execution time observed so far and then select the slowest query to tune next.

3.2.3 Multi-armed Bandit

This strategy models workload tuning as a multi-armed bandit problem. Specifically, we view each query as an arm, and use the well-known UCB1 score [2, 3] as the criterion for selecting the next query to tune:

$$\operatorname{argmax}_q \left[\bar{r}(q) + \lambda \cdot \sqrt{\frac{\ln N}{f_q}} \right].$$

Here, λ is a constant that balances *exploration* and *exploitation*. We choose $\lambda = \sqrt{2}$ as suggested in the literature [17]. f_q is the number of times (i.e. frequency) that query q is tuned, and $N = \sum_{q \in W} f_q$. $\bar{r}(q)$ is the *average reward* of q . The reward $r(q, \vec{u})$ of tuning q with cost units \vec{u} is defined as its relative improvement over the query execution time with the default cost units \vec{u}_0 : $r(q, \vec{u}) = \max\{1 - \frac{t(q, \vec{u})}{t(q, \vec{u}_0)}, 0\}$. That is, we cap the reward at 0 if \vec{u} is even worse than \vec{u}_0 . This should not occur if the early-stopping optimization is used. The average reward is therefore $\bar{r}(q) = \frac{1}{f_q} \sum_{j=1}^{f_q} r(q, \vec{u}_j)$. Again, we stop when the tuning time budget is exhausted.

3.2.4 Improvement Rate

The improvement $I_j(q)$ of a query q is defined as the gap between its best execution time found so far and the default execution time with cost units \vec{u}_0 . That is, $I_j(q) = \max\{t(q, \vec{u}_0) - t(q, \vec{u}_j^*), 0\}$, for $1 \leq j \leq f_q$. Again, we cap the improvement at 0 if \vec{u}_j^* is worse than \vec{u}_0 , which should not happen if the early-stopping optimization is used. The improvement rate is then defined as $R_j(q) = \frac{I_j(q)}{f_q}$. This strategy always selects the query with the highest improvement rate to tune next, which is inspired by BlendSearch [30]. Again, we stop when the tuning time budget is exhausted.

3.2.5 Summary and Discussion

The strategies discussed above are by no means perfect or exhaustive. We briefly discuss potential improve-

Name	DB Size	Queries	Tables	Joins	Scans
JOB	9.2GB	33	21	7.9	8.9
TPC-DS	<i>sf=10</i>	99	24	7.7	8.8
Real-A	100GB	25	20	6.5	7.2
Real-B	60GB	16	7	1.9	2.9

Table 2: Database and workload statistics.

ments, extensions, and other alternatives. First, it is not necessary to always start running these strategies from scratch. As we mentioned, there has been prior work on calibrating cost units (e.g., [11,26,35]) by viewing them as constants. These refined cost constants can be used as starting points of the above strategies for further fine-tuning. Second, the cost-unit calibration technologies could themselves serve as solutions to the WT problem, especially for its query-independent variant (see Definition 3), though in a budget-unaware sense. However, these technologies typically come with additional implementation overhead as well as subtleties that the WT strategies proposed in this paper do not have. For example, one technique used in [35] is to design a set of *independent* “calibration queries” that can target individual cost units. This is relatively easy for some database systems such as PostgreSQL but becomes more challenging for others such as Microsoft SQL Server. Not only does Microsoft SQL Server have many more cost units compared to PostgreSQL, but some of its cost units are also *correlated*. Since no calibration query can separate two correlated cost units, the technique from [35] needs to be extended, which requires further research.

4. EXPERIMENTAL EVALUATION

We report experimental results on evaluating the performance of our proposed budget-aware query and workload tuning technologies.

Datasets and Workloads. We used various benchmark and real workloads in our evaluation. For benchmark workloads, we use the join order benchmark (**JOB**) [18], as well as the **TPC-DS** benchmark with scaling factor 10. **JOB** contains 113 query instances in total, which are grouped into 33 templates, and we pick one query instance from each template. We use the same protocol for **TPC-DS**. We choose these two benchmark workloads due to the diversity and complexity in their queries that offer more opportunities for finding better query plans via query tuning. We also use two real workloads, denoted by **Real-A** and **Real-B**. Table 2 summarizes the key statistics of these workloads. The last two columns represent the average number of joins and table scans contained by a query in the workload.

Experimental Settings. We perform all experiments using Microsoft SQL Server 2017 under Windows Server 2022, running on a workstation equipped with 2.3 GHz AMD CPUs and 256 GB main memory. We focus on tuning eight cost units that are critical to the costs of workhorse operators such as *table scan*, *index seek*, *sort*,

Name	Default Plan (minutes)	Best Plan (minutes)	Percentage Improvement (%)	Tuning Time (minutes)
JOB	3.72	1.33	64.2%	130
TPC-DS	4.74	3.24	31.6%	580
Real-A	13.96	7.36	47.3%	675
Real-B	13.21	8.38	36.5%	160

Table 3: Summary of query tuning results

hash join, and *nested-loop join*. For each cost unit c , we set its range/domain as $[0.1 \times c_d, 10 \times c_d]$ for exploration (ref. Section 2.1), where c_d is the default value of c .

4.1 Query Tuning Results

We use *percentage improvement* as the performance metric, defined as $1 - \frac{t(P_{\text{best}})}{t(P_{\text{default}})}$. $t(P_{\text{best}})$ and $t(P_{\text{default}})$ represent the execution time of the best plan found by QT and that of the default plan chosen by the query optimizer (using the built-in values of the cost units).

For each query, we give the HPO algorithm \mathcal{A} 100 trials, and report the best plan found. For the HPO algorithm \mathcal{A} , we evaluated both *random search* [5] and *SMAC* [16], but we found that their performances were similar. As a result, we only report results by using random search, due to its simplicity and lower overhead.

Table 3 summarizes the QT results for the workloads evaluated. We observe percentage improvement ranging from 31.6% to 64.2% at workload level (i.e., the total execution time of all queries). Figure 1 further showcases the percentage improvement for each query in the **JOB** workload, whereas Figure 2 presents the corresponding execution time (in seconds) of the default plan and the best plan found by QT. Note that we did not test the budget-aware version of QT, as it is a special case of budget-aware WT that will be covered in Section 4.2.

Case Study. We further present a case study of the **JOB** query #5, which shows significant improvement in Figure 2. We observe that the most significant parameter changes when tuning this query happen in cost units related to random reads and index seeks: the values of these cost units are significantly decreased (by $5\times$) for the best plan found by QT. Since the **JOB** database size (ref. Table 2) is much smaller than the server’s memory size (i.e., 256 GB), most of the database can be cached in the main memory. As a result, random reads and index seeks are much faster compared to the case when the database resides on disk. This sheds some light on the significant improvement observed for this query.

Discussion. From Table 3, the improvements across workloads vary. Clearly, the improvement is determined by the gap between $t(P_{\text{default}})$ and $t(P_{\text{best}})$. One important factor that contributes to this gap is query complexity: for a simple query with no joins, the gap is likely small; for a complex query with many joins, the gap is likely large. Meanwhile, the degree of cardinality estimation (CE) errors also matters, as it is well-known

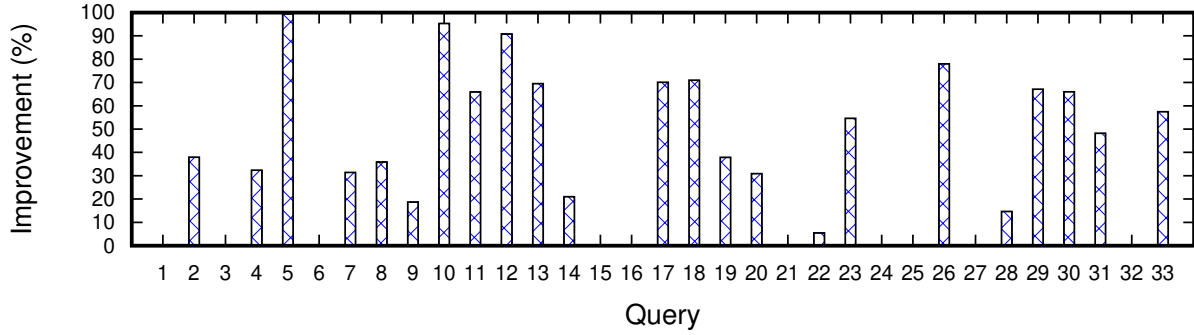


Figure 1: Percentage improvement of each query in the JOB workload.

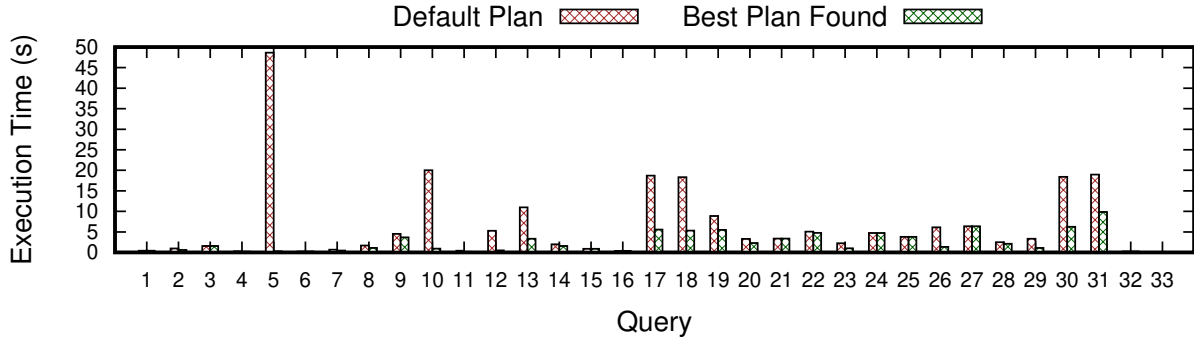


Figure 2: Execution time of default plan vs. best plan found by QT for each query of the JOB workload.

that query optimizers are likely to choose poor execution plans in the presence of CE errors [24]. Hence, we would expect larger improvements for workloads with complex queries and significant CE errors. From the results reported in Table 3, we observe the largest percentage improvement on the **JOB** workload. This does not seem like a coincident, as **JOB** is intentionally designed to challenge the cardinality estimators of query optimizers with complex join queries (ref. Table 2) [19].

4.2 Workload Tuning Results

We vary the budget on the time given to the workload tuning algorithms and test the percentage improvement at workload-level. Figure 3 presents the results on the workloads tested. The x -axis represents the tuning time given to the algorithms, whereas the y -axis reports the percentage improvement observed. To avoid clutter, Figure 3 only includes the two best-performing algorithms, *round robin* and *multi-armed bandit*, which significantly outperform the other two algorithms, *cost-based prioritization* and *improvement rate* (see Figure 4 for a comparison on **TPC-DS**). Moreover, the dashed line in each chart represents the percentage improvement observed in the QT experiment with 100 trials of random search for each query, whereas the corresponding tuning time has been reported in the last column of Table 3. We observe that we can obtain similar percentage improvement with much less tuning time. For example, on **JOB** it took only 80 minutes for both algorithms to achieve the same percentage improvement, compared to the 130 minutes in QT (i.e., 38.5% reduc-

tion); on **Real-A**, it took only 98 minutes for *round robin* to achieve the same percentage improvement, in contrast to the 675 minutes taken in QT (i.e., 85.5% reduction).

Comparison of Workload Tuning Algorithms. Figure 4 compares the four workload tuning algorithms with varying tuning time budget, using the **TPC-DS** workload. We observe that *round robin* and *multi-armed bandit* perform similarly, and they significantly outperform the other two algorithms, *cost-based prioritization* and *improvement rate*. The *cost-based prioritization* strategy often does not perform well in a budget-aware setting, because it can often get stuck on some very expensive query that is also hard to improve. The *improvement rate* strategy bypasses this issue by focusing on tuning queries that can improve quickly. As a result, it improves over the *cost-based prioritization* strategy. However, it remains less effective compared to *round robin* and *multi-armed bandit*. On the other hand, it is somewhat surprising to see *round-robin* performs closely to the best-performing *multi-armed bandit* strategy in most of the cases. One reason could be that, for the workloads that we tested, all queries are worth tuning and there is little need to prioritize. However, we do not expect that this property holds in general, and we expect that the *multi-armed bandit* strategy should outperform *round-robin* for workloads with more heterogeneous queries. The question of designing a better strategy for budget-aware workload tuning (other than the ones studied in this paper, which are somewhat standard) remains open and we intend to leave it for future work.

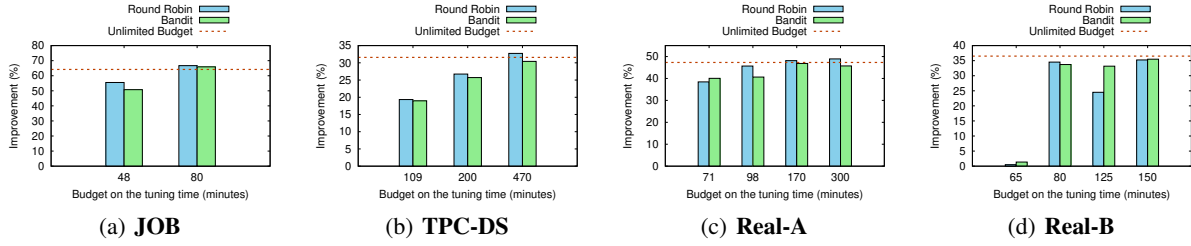


Figure 3: Percentage improvement resulted from workload tuning when varying the budget on tuning time.

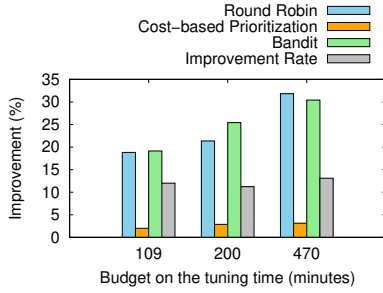


Figure 4: Comparison of WT algorithms on TPC-DS

5. RELATED WORK

Autonomous Knob Tuning. The problem of knob tuning for database systems has attracted intensive research interest (see [40] for a recent survey on this topic). Some existing work, such as UDO [31], also views the query optimizer cost units as tunable knobs. Our view is that the cost units are different from other runtime configuration knobs that change the database server’s runtime state. The main impact of the cost units is to influence the query optimizer to generate different query plans.

Autonomous Index Tuning. While some existing work also considers index tuning as part of knob tuning (e.g., UDO [31]), we do think that it falls into another special category of database tuning, just like tuning the cost units, which is worth its own treatment. The reason is similar—index tuning has a different impact on the database system compared to knobs that can change the database server’s runtime state. Specifically, index tuning will change the physical data layout of the database, which in some sense is more dramatic as it has broader impact on various database system components, such as metadata management, query optimizer’s plan choice, statistics (such as histograms) maintenance, and so on. We refer the readers to [28] for a recent survey on index tuning. The idea of budget-aware index tuning has also been explored recently [32, 37], with the similar motivation of constraining the tuning time in practice.

Hyper-parameter Optimization. The HPO problem has been extensively studied in the literature (see [39] for a survey). In addition to simple strategies such as *random search* [5] and bandit-based strategies such as Hyperband [20], many HPO strategies rely on classic Bayesian Optimization (BO), such as Hyperopt [4, 6], SMAC [16, 23], Spearmint [29], BOHB [12], and Open-

Box [22]. These BO-style HPO strategies view the function from the hyper-parameter values to the ML model quality metric (e.g., accuracy) as a *black box* without utilizing workload properties. While in this paper we use these off-the-shelf HPO strategies without modification, an interesting future direction to explore is to leverage the similarity among the workload queries [10, 27] and use that information to improve BO-style strategies.

6. CONCLUSION

In this paper, we proposed the budget-aware query tuning and workload tuning problems. We highlighted the connection between the query tuning problem and the hyper-parameter optimization (HPO) problem in AutoML, and we proposed solutions based on adapting existing HPO algorithms such as *random search* and *SMAC*. Experimental evaluation shows that (1) query tuning can result in much faster query plans compared to the ones generated by the query optimizer based on the default values of the cost units; and (2) budget-aware workload tuning using simple strategies such as *round robin* or *multi-armed bandit* can significantly reduce the amount of tuning time at workload-level.

We note that the query and workload tuning technologies proposed in this paper are rudimentary as they are simple applications of existing well-known technologies. As a result, they should serve as baseline approaches that future research can reference and compare against. One promising direction for future work, as we briefly mentioned, is to further leverage the similarities among workload queries to improve the BO-style approaches. For instance, a particular set of cost-unit values may be optimal for multiple queries if they share common SQL expressions. As a result, one may want to perform a clustering on the queries and tune each group/cluster of queries as an independent, smaller workload. This can further reduce the tuning time on a large workload, or can have more potential of finding better query plans within a given tuning time budget. Nonetheless, it also raises new challenges such as how to cluster the queries and how to prioritize among the query clusters during workload tuning, which requires further investigation.

Acknowledgement. We thank the anonymous reviewers, Anshuman Dutt, Bailu Ding, and Vivek Narasayya for their valuable feedback on this work.

7. REFERENCES

- [1] PostgreSQL planner cost constants. <https://www.postgresql.org/docs/current/runtime-config-query.html>, 2024.
- [2] P. Auer. Using confidence bounds for exploitation-exploration trade-offs. *J. Mach. Learn. Res.*, 3:397–422, 2002.
- [3] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2-3):235–256, 2002.
- [4] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *NIPS*, 2011.
- [5] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, 2012.
- [6] J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *ICML*, volume 28, pages 115–123, 2013.
- [7] N. Bruno, S. Chaudhuri, and R. Ramamurthy. Power hints for query optimization. pages 469–480, 2009.
- [8] J. Chang. Sql server query optimizer cost formulas. <https://slidetodoc.com/sql-server-query-optimizer-cost-formulas-joe-chang-3/>, 2010.
- [9] S. Chaudhuri and V. Narasayya. Anytime algorithm of database tuning advisor for microsoft sql server, June 2020.
- [10] S. Deep, A. Gruenheid, P. Koutris, J. F. Naughton, and S. Viglas. Comprehensive and efficient workload compression. *Proc. VLDB Endow.*, 14(3):418–430, 2020.
- [11] W. Du, R. Krishnamurthy, and M. Shan. Query optimization in a heterogeneous DBMS. In *VLDB*, pages 277–291, 1992.
- [12] S. Falkner, A. Klein, and F. Hutter. BOHB: robust and efficient hyperparameter optimization at scale. In *ICML*, volume 80, pages 1436–1445, 2018.
- [13] X. He, K. Zhao, and X. Chu. Automl: A survey of the state-of-the-art. *Knowl. Based Syst.*, 212:106622, 2021.
- [14] H. Herodotou and S. Babu. Xplus: A sql-tuning-aware query optimizer. *Proc. VLDB Endow.*, 3(1):1149–1160, 2010.
- [15] R. Hu, Z. Wang, W. Fan, and S. Agarwal. Cost based optimizer in apache spark 2.2. <https://www.databricks.com/blog/2017/08/31/cost-based-optimizer-in-apache-spark-2-2.html>, 2017.
- [16] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *LION*, volume 6683, pages 507–523, 2011.
- [17] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *ECML*, pages 282–293, 2006.
- [18] V. Leis. Join order benchmark. <https://github.com/gregrahn/join-order-benchmark>.
- [19] V. Leis, A. Gubichev, A. Mirchev, P. A. Boncz, A. Kemper, and T. Neumann. How good are query optimizers, really? *Proc. VLDB Endow.*, 9(3):204–215, 2015.
- [20] L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.*, 18:185:1–185:52, 2017.
- [21] T. Li, J. Zhong, J. Liu, W. Wu, and C. Zhang. Ease.ml: Towards multi-tenant resource sharing for machine learning workloads. *Proc. VLDB Endow.*, 11(5):607–620, 2018.
- [22] Y. Li, Y. Shen, W. Zhang, Y. Chen, H. Jiang, M. Liu, J. Jiang, J. Gao, W. Wu, Z. Yang, C. Zhang, and B. Cui. Openbox: A generalized black-box optimization service. In *KDD*, pages 3209–3219, 2021.
- [23] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter. SMAC3: A versatile bayesian optimization package for hyperparameter optimization. *JMLR*, 23, 2022.
- [24] G. Lohman. Is query optimization a “solved” problem? <http://wp.sigmod.org/?p=1075>.
- [25] G. M. Lohman. The db2 universal database optimizer. <https://cs.uwaterloo.ca/ilyas/CS448W14/ibm.pdf>, 2003.
- [26] L. F. Mackert and G. M. Lohman. R* optimizer validation and performance evaluation for distributed queries. In *VLDB*, pages 149–159, 1986.
- [27] T. Siddiqui, S. Jo, W. Wu, C. Wang, V. R. Narasayya, and S. Chaudhuri. ISUM: efficiently compressing large and complex workloads for scalable index tuning. In *SIGMOD*, pages 660–673. ACM, 2022.
- [28] T. Siddiqui and W. Wu. ML-powered index tuning: An overview of recent progress and open challenges. *SIGMOD Rec.*, 52(4):19–30, 2023.
- [29] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *NIPS*, 2012.
- [30] C. Wang, Q. Wu, S. Huang, and A. Saied. Economic hyperparameter optimization with blended search strategy. In *ICLR*, 2021.
- [31] J. Wang, I. Trummer, and D. Basu. UDO: universal database optimization using reinforcement learning. *Proc. VLDB Endow.*, 14(13):3402–3414, 2021.
- [32] X. Wang, W. Wu, C. Wang, V. Narasayya, and S. Chaudhuri. Wii: Dynamic budget reallocation in index tuning. *Proc. ACM Manag. Data*, 2(3), 2024.
- [33] Q. Wu, C. Wang, and S. Huang. Frugal optimization for cost-related hyperparameters. In *AAAI*, 2021.
- [34] W. Wu, Y. Chi, H. Hacigümüs, and J. F. Naughton. Towards predicting query execution time for concurrent and dynamic database workloads. *PVLDB*, 6(10):925–936, 2013.
- [35] W. Wu, Y. Chi, S. Zhu, J. Tatemura, H. Hacigümüs, and J. F. Naughton. Predicting query execution time: Are optimizer cost models really unusable? In *ICDE*, pages 1081–1092, 2013.
- [36] W. Wu, J. F. Naughton, and H. Singh. Sampling-based query re-optimization. In F. Özcan, G. Koutrika, and S. Madden, editors, *SIGMOD*, pages 1721–1736. ACM, 2016.
- [37] W. Wu, C. Wang, T. Siddiqui, J. Wang, V. R. Narasayya, S. Chaudhuri, and P. A. Bernstein. Budget-aware index tuning with reinforcement learning. In Z. G. Ives, A. Bonifati, and A. E. Abbadi, editors, *SIGMOD*, pages 1528–1541, 2022.
- [38] W. Wu, X. Wu, H. Hacigümüs, and J. F. Naughton. Uncertainty aware query execution time prediction. *PVLDB*, 7(14):1857–1868, 2014.
- [39] T. Yu and H. Zhu. Hyper-parameter optimization: A review of algorithms and applications. *CoRR*, abs/2003.05689, 2020.
- [40] X. Zhao, X. Zhou, and G. Li. Automatic database knob tuning: A survey. *IEEE Trans. Knowl. Data Eng.*, 35(12), 2023.

Reminiscences on Influential Papers

This issue's contributors dive into a history of concurrency control and storage hierarchy for database and data-intensive systems. Enjoy reading!

While I will keep inviting members of the data management community, and neighboring communities, to contribute to this column, I also welcome unsolicited contributions. Please contact me if you are interested.

Pinar Tözün, *editor*
IT University of Copenhagen, Denmark
pito@itu.dk

Goetz Graefe
Google - Madison, WI, US
goetzg@google.com

H. T. Kung and John T. Robinson.

On optimistic methods for concurrency control.

In ACM Transactions on Database Systems (TODS), Volume 6, Issue 2, pages 213-226, 1981.

This fall I will teach “topics in database systems” at UW-Madison. When I took this course as a first-year graduate student four decades ago, David DeWitt had his students read two then-recent papers that he correctly expected to become classics: “Access Path Selection...” [29] and “On Optimistic... Concurrency Control” [21]. Of these, the former still strikes me as brilliant for constructing join plans using dynamic programming, for its focus on “interesting orderings” in complex-object assembly (joining and grouping on, effectively, object identifiers), and for its clear-eyed perspective on the perils of cardinality estimation. Kooi’s impressive thesis soon addressed some of these perils with multiple kinds of histograms [20].

The latter one of these two classic papers res-

onates with the saying that “it’s easier to ask for forgiveness than for permission” but clearly there is much more to it. Many people, it sometimes seems, remember the paper’s title but have not really studied the text. They may rely on the title’s promise but ignore multiple pitfalls. Many pitfalls have been called out by Theo Härder and C. Mohan [19, 25] – below is one further perspective.

First, the original design of optimistic concurrency control assumes page-level concurrency control. This was competitive at the time, e.g., with page-level locking in System R [8], but it is not competitive with record-level locking [22, 24, 27]. The introduction of record-level concurrency required new techniques for logging and recovery, e.g., compensation or update-back log records [14, 26], and for concurrency control, e.g., locking gaps between index entries or between key values [13]. More specifically, equivalence to a serial execution requires repeatable read plus phantom protection [9]. For example, in an ordered database index, a search with an empty result set requires concurrency control for gaps between key values, as does a range scan.

Second, optimistic concurrency control requires transaction-private update buffers, which are a form of multi-version storage. A fair comparison of optimistic and pessimistic concurrency control (i.e., end-of-transaction validation versus pre-access locking) must consider locking in a multi-version context, not in single-version storage. In a multi-version store, locking can ignore read-write (rw-) conflicts and wr-conflicts until the updating transaction attempts to commit. Detecting ww-conflicts upon the first conflicting access prevents conflicting writers and doomed transactions, saves wasted work and its rollback, ensures at most one uncommitted version (per granule of concurrency control), and permits creating new versions directly in the database (or its shared buffer pool) - all with the hope and expectation that rw- and wr-conflicts rarely delay transaction commit. This optimism is the founda-

tion and essence of deferred lock enforcement [12].

Third, validation and write phases together must be atomic, as described in the original 1981 paper. For transactional durability [18], the write phase is more than draining a transaction's private update buffer into the system's shared buffer pool: each transaction's write phase must include forcing the commit log record to stable storage. With fairly long atomic validation-and-write phases, shared database servers require concurrent validation. Transactions validating concurrently must share information about their read- and write-sets. The required data structure manages, in a thread-safe manner, a many-to-many relationship between transactions and database objects - very much like a traditional lock manager. In this way of thinking, optimistic concurrency control is a form of deferred lock acquisition [12]. Its expectations for performance, scalability, conflicts, and system throughput are near but not quite equal to deferred lock enforcement.

Fourth, participants of distributed transactions transition from optimism to pessimism when they vote in a two-phase commit. More accurately, they pledge to wait for and to implement the coordinator's global commit decision. This pledge must be firm; "asking for forgiveness" is no longer an option so that "asking for permission" is required, i.e., lock acquisition. Again, optimistic concurrency control is a form of deferred lock acquisition. Participants must retain both shared and exclusive locks until they receive the global commit decision [11]. Increasing concurrency during the commit process beyond the traditional level has been described as controlled lock violation [15]. Other transactions may read and modify recent updates but they must not commit until the commit logic of the earlier updater completes successfully. In a sense, controlled lock violation is an optimistic technique within pessimistic concurrency control.

In summary, optimism in concurrency control is found in both deferred lock enforcement and controlled lock violation, which are advanced locking techniques with optimism, with perfect equivalence to a serial execution, with early detection of doomed transactions, with no need or overhead for transaction-private update buffers, and with fewer false conflicts than traditional optimistic or pessimistic concurrency control. Even if these locking techniques supplant the original design for optimism in concurrency control, they are continuations and refinements of the classic 1981 paper on optimistic concurrency control.

Raja Appuswamy

EURECOM, France

raja.appuswamy@eurecom.fr

Jim Gray and Franco Putzolu.

The 5 minute rule for trading memory for disc accesses and the 10 byte rule for trading memory for CPU time.

In Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data, pages 395–398, 1987.

In 1998, Richard Snodgrass edited a new column in SIGMOD Record entitled "Reminiscences on Influential Papers". In the introduction to this column, he wrote "The researcher comes across a paper that touches something deep inside, triggering a radical restructuring of their mental model and allowing them to see things in a new light." When Pinar reached out to me for a contribution, for which I am very thankful, I instantly knew which paper I had to choose.

Over the years of my research career, I have worked on several systems that rely on caching at various levels of the memory hierarchy. I had always viewed caching from an algorithmic point of view. The most interesting part in caching, I had thought, was the design of clever algorithms that could identify the optimal data items to cache. I was fascinated by the simplicity of design and the incredible potential of the Adaptive Replacement Cache [23] when it was introduced by IBM and adopted by ZFS, and toyed around with extending it to a multi-level memory hierarchy [4] during my PhD years when I designed the Loris file system for MINIX 3. And then, I read the five-minute rule paper, and as Richard said in 1998, it definitely triggered a radical restructuring of my mental model.

Jim Gray and Franco Putzolu introduced the five-minute rule in 1987 [17]. In a surprising twist, Jim and Franco turned the caching problem into an economic one. They argued that bringing disk-resident data to memory is not only time consuming, but is also economically expensive. At the time, the Tandem disk cost \$15K and could provide 15 accesses per second. Thus, it cost \$1K per access per second. Add in support for CPUs to handle interrupts and channel/controller costs to manage the disks, you get to \$2K per access per second. A Kilobyte of main memory, in contrast, costs only \$5. So, assuming you have 1KB of data that you access every second, if you "rent" 1KB of memory and keep your data in it, you can save \$2K worth of disk

accesses—a fantastic bargain. If you access the 1KB every 10 seconds, that is 0.1 accesses/second, you can still save \$200 of disk access by spending \$5 to rent memory. Continuing this argument, it turns out that accessing data once every 400 seconds becomes the break-even point, when accessing data from disk is as expensive as renting memory. Restating Jim and Franco, “as 400 seconds is about five minutes, hence the name five-minute rule”.

I love this paper for several reasons. First, it is written by one of the titans of computing I respect deeply—Jim Gray. My first encounter with Jim’s work was the seminal tome on transaction processing. While writing this article, I found out that Jim himself had contributed an article to this very column back in 1998, where he describes the story of how TPC-A and TPC-D benchmarks came about, and why SIGMOD sort trophies used to be awarded on April Fools Day! His description of the memory hierarchy in the popular “How Far Away is the Data” example is yet another masterpiece like the five-minute rule that breaks down a complex topic into a fun, easy-to-understand exposition that I routinely use in my lectures.

Second, in the paper, Jim and Franco provide a quantitative framework for making informed decisions about memory and disk usage. Using a case study, they even show how the five-minute rule was used in practice to help a designer choose an appropriate configuration (hybrid memory-disk versus all-in-memory) for a database server. Storage technology and economics have changed radically since the rule was introduced in 1987. Yet, the framework proposed in the original paper is general enough to apply to various levels of today’s three-tier (performance with DRAM/SSD/PMEM, capacity with HDD, and archival with tape) memory hierarchy. Thus, the rule has been revisited three times, in 1997 [16], 2008 [10], and 2019 [3], with each iteration providing surprising insights into the behavior of “media-du-jour”, be it NAND flash in 2008 or persistent memory in 2019.

Third, this paper really made me think about how other aspects of data management in addition to caching would change if we considered economics, and cost of data engineering, as first class citizens instead of performance. This made me focus my research over the past few years on the lower two tiers of the storage hierarchy. Focusing on the capacity tier, I was inspired by Pelican [5]—a rack-scale cold storage system that packed thousands of HDDs in a single rack that was right-provisioned to service only a fraction of HDDs running simultaneously. Pelican provided accesses latencies in sec-

onds, between HDD and tape, and provided near-line data access for “cold”, or infrequently accessed, data items. Applying the five-minute rule framework to such cold storage devices (CSD) showed us that it might be economically beneficial to leave cold data in such devices and perform query execution directly on the CSD rather than moving it to HDD. This motivated us to develop the Skipper query-processing framework [7]. Focusing on the archival tier, I came across the work on database preservation by the Digital Preservation community that made me realize that long-term archival of databases is an incredibly challenging problem, both from technical and economic points of view. I wrote about some of these challenges in the SIGMOD Record Brainstorming article [1] “Towards Passive, Migration-Free, Standardized, Long-Term Database Archival”. Around this time, storage researchers were starting to investigate radically new archival media, and synthetic DNA was one such media that got a lot of attention [6]. This triggered our work on project OligoArchive [2], where we investigated the use of DNA for long-term database archival.

Fourth, the paper’s history teaches an important life lesson for academics: rejection, just like awards in some cases, is not a concrete predictor of impact. Despite being such an insightful piece of work, this paper was rejected during its first round of submission. Quoting David Patterson [28], “Jim Gray wrote to Jim Larus: The B-tree paper was rejected at first. The Transaction paper was rejected at first. The data cube paper was rejected at first. The five-minute rule paper was rejected at first. But linear extensions of previous work get accepted. So, re-submit! PLEASE!!.” I have had my papers rejected enough times now to know and tell my students that it is a part of academic life. But using THE five-minute rule from a titan such as Jim Gray as an example of rejection usually sends home the message right away.

1. REFERENCES

- [1] Raja Appuswamy. Towards Passive, Migration-Free, Standardized, Long-Term Database Archival. *SIGMOD Rec.*, 51(2):61–62, jul 2022.
- [2] Raja Appuswamy, Kevin Le Brigand, Pascal Barbry, Marc Antonini, Olivier Madderson, Paul S. Freemont, James McDonald, and Thomas Heinis. OligoArchive: Using DNA in the DBMS storage hierarchy. In *9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA*,

- January 13-16, 2019, *Online Proceedings*. www.cidrdb.org, 2019.
- [3] Raja Appuswamy, Goetz Graefe, Renata Borovica-Gajić, and Anastasia Ailamaki. The five-minute rule 30 years later and its impact on the storage hierarchy. *Commun. ACM*, 62(11):114–120, oct 2019.
 - [4] Raja Appuswamy, David C. van Moelenbroek, and Andrew S. Tanenbaum. Cache, cache everywhere, flushing all hits down the sink: On exclusivity in multilevel, hybrid caches. In *2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–14, 2013.
 - [5] Shobana Balakrishnan, Richard Black, Austin Donnelly, Paul England, Adam Glass, Dave Harper, Sergey Legtchenko, Aaron Ogus, Eric Peterson, and Antony Rowstron. Pelican: A building block for exascale cold data storage. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation, OSDI’14*, page 351–365, USA, 2014. USENIX Association.
 - [6] James Bornholt, Randolph Lopez, Douglas M. Carmean, Luis Ceze, Georg Seelig, and Karin Strauss. A DNA-Based Archival Storage System. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS ’16*, page 637–649, New York, NY, USA, 2016. Association for Computing Machinery.
 - [7] Renata Borovica-Gajić, Raja Appuswamy, and Anastasia Ailamaki. Cheap data analytics using cold storage devices. *Proc. VLDB Endow.*, 9(12):1029–1040, aug 2016.
 - [8] Donald D. Chamberlin, Morton M. Astrahan, Michael W. Blasgen, James N. Gray, W. Frank King, Bruce G. Lindsay, Raymond Lorie, James W. Mehl, Thomas G. Price, Franco Putzolu, Patricia Griffiths Selinger, Mario Schkolnick, Donald R. Slutz, Irving L. Traiger, Bradford W. Wade, and Robert A. Yost. A history and evaluation of System R. *Commun. ACM*, 24(10):632–646, oct 1981.
 - [9] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger. The notions of consistency and predicate locks in a database system. *Commun. ACM*, 19(11):624–633, nov 1976.
 - [10] Goetz Graefe. The five-minute rule 20 years later (and how flash memory changes the rules). *Commun. ACM*, 52(7):48–59, jul 2009.
 - [11] Goetz Graefe. *On Transactional Concurrency Control: A Problem in Two-Phase Commit*, pages 321–326. Springer International Publishing, Cham, 2019.
 - [12] Goetz Graefe. *On Transactional Concurrency Control: Deferred Lock Enforcement*, pages 327–365. Springer International Publishing, Cham, 2019.
 - [13] Goetz Graefe. *On Transactional Concurrency Control: Orthogonal Key-Value Locking*, pages 159–210. Springer International Publishing, Cham, 2019.
 - [14] Goetz Graefe, Wey Guy, and Caetano Sauer. *Instant Recovery with Write-Ahead Logging: Page Repair, System Restart, Media Restore, and System Failover, Second Edition*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2016.
 - [15] Goetz Graefe, Mark Lillibridge, Harumi Kuno, Joseph Tucek, and Alistair Veitch. Controlled lock violation. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, SIGMOD ’13*, page 85–96, New York, NY, USA, 2013. Association for Computing Machinery.
 - [16] Jim Gray and Goetz Graefe. The five-minute rule ten years later, and other computer storage rules of thumb. *SIGMOD Rec.*, 26(4):63–68, dec 1997.
 - [17] Jim Gray and Franco Putzolu. The 5 minute rule for trading memory for disc accesses and the 10 byte rule for trading memory for CPU time. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data, SIGMOD ’87*, page 395–398, New York, NY, USA, 1987. Association for Computing Machinery.
 - [18] Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM Comput. Surv.*, 15(4):287–317, dec 1983.
 - [19] Theo Härder. Observations on optimistic concurrency control schemes. *Information Systems*, 9(2):111–120, 1984.
 - [20] Robert Philip Kooi. *The optimization of queries in relational databases*. PhD thesis, USA, 1980. AAI8109596.
 - [21] H. T. Kung and John T. Robinson. On optimistic methods for concurrency control. *ACM Trans. Database Syst.*, 6(2):213–226, jun 1981.
 - [22] David B. Lomet. Key Range Locking Strategies for Improved Concurrency. In *Proceedings of the 19th International Conference on Very Large Data Bases, VLDB ’93*, page 655–664, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

- [23] Nimrod Megiddo and Dharmendra S. Modha. ARC: A Self-Tuning, Low Overhead Replacement Cache. In *2nd USENIX Conference on File and Storage Technologies (FAST 03)*, San Francisco, CA, March 2003. USENIX Association.
- [24] C. Mohan. ARIES/KVL: A key-value locking method for concurrency control of multi-action transactions operating on B-tree indexes. In *Proceedings of the Sixteenth International Conference on Very Large Databases*, page 392–405, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.
- [25] C. Mohan. Less optimism about optimistic concurrency control. In *[1992 Proceedings] Second International Workshop on Research Issues on Data Engineering: Transaction and Query Processing*, pages 199–204, 1992.
- [26] C. Mohan, Don Haderle, Bruce Lindsay, Hamid Pirahesh, and Peter Schwarz. ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM Trans. Database Syst.*, 17(1):94–162, mar 1992.
- [27] C. Mohan and Frank Levine. ARIES/IM: An efficient and high concurrency index management method using write-ahead logging. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, SIGMOD '92, page 371–380, New York, NY, USA, 1992. Association for Computing Machinery.
- [28] David A. Patterson. How to Build a Bad Research Center. Technical Report UCB/EECS-2013-123, Jun 2013.
- [29] P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, SIGMOD '79, page 23–34, New York, NY, USA, 1979. Association for Computing Machinery.

The Road to Explainable Graph Neural Networks

Sayan Ranu

Department of Computer Science & Engineering, and Yardi School of AI (Joint Appointment), IIT Delhi
sayanranu@iitd.ac.in

GOAL: Graph Neural Networks (GNNs) are being increasingly adopted in various real-world applications, including drug and material discovery [16, 17, 27], recommendation engines [29], congestion modeling on road networks [5], and weather forecasting [11]. However, similar to other deep-learning models, GNNs are considered black boxes due to their limited capacity to provide explanations for their predictions. This lack of interpretability poses a significant barrier to their adoption in critical domains such as healthcare, finance, and law enforcement, where transparency and trustworthiness are essential for decision-making processes. In these pivotal domains, understanding the rationale behind model predictions is crucial not only for compliance with interpretability requirements but also for identifying potential vulnerabilities and gaining insights to refine the model further. *How do we make GNNs interpretable?* This is a central motivating question driving my research pursuits.

BACKGROUND: Existing GNN explainers can be grouped into *model-level* and *instance-level* explainers. Model-level explainers [8, 26, 32] aim to explain the overall behavior of the model that generalize across instances. Instance-level explainers [1, 3, 7, 12, 13, 14, 19, 21, 25, 30, 33, 34] explain a specific input graph by highlighting the subgraph components used by a GNN to make its prediction. The type of explanation offered is either *factual* [15, 24, 28, 32] or *counterfactual* [2, 4, 13, 22]. Factual explainers aim to find an important subgraph that correlates most with the underlying GNN prediction. In contrast, counterfactual reasoning attempt to identify the smallest amount of perturbation on the input graph (e.g., removal/addition of edges or nodes) that changes the GNN’s prediction. Thus, while factual explainers surface factors influencing a prediction, counterfactual explainers provide insights into how small changes in the input lead to different outcomes.

CHALLENGES AND IDEAS:

Completeness: Factual explanations aim to identify the cause of a particular prediction. However, an interesting observation has emerged: when the explanations (subgraph) are removed from the input graphs, and the GNN is retrained on the residual graphs, it often manages to recover the correct predictions [9, 10]. This observation prompts a crucial inquiry: *Are the explanations complete?* The incompleteness

of explainers is likely a manifestation of their *post-hoc* learning framework. Specifically, the post-hoc paradigm treats the GNN as a black box where the explainers have no visibility to model internals such as the loss surface and hyper-parameters.

Idea: Remedies may lie within the *ante-hoc* paradigm, where the GNN and the explainer undergo *joint* training [9]. Joint training enables access to a more comprehensive set of information influencing the GNN predictions, including the loss surface, topological components of the input activating neurons, and gradient flows through the layers.

Feasibility: An important facet of counterfactual reasoning lies in its ability to offer recourse options. The practical effectiveness of the recourse hinges on its alignment with specific domain constraints. In molecular datasets, for instance, a valid recourse should yield a chemically sound molecule. Current counterfactual explainers predominantly focus on pinpointing the shortest edit path that steers the graph toward the decision boundary, often overlooking the feasibility of the proposed edits [10].

Idea: The loss associated with a counterfactual explanation is typically a function of: (1) the size of the explanation and (2) the quality of the explanation. I propose the addition of a third term that models the *feasibility* of the explanation. Modeling feasibility presents non-trivial challenges, and potential solutions may be found in the area of generative modeling for graphs [6, 23, 31]. Generative models learn the underlying distribution of topological patterns in the training data and utilize that knowledge to generate new, similar graphs. This paradigm can be extended to estimate the probability of the recourse being generated, corresponding to the feasibility term in the loss function.

Stability: Humans are the intended audience for explanations. Hence, they need to be stable. Explainers are neural networks themselves optimizing parameters on a non-convex loss surface. Consequently, they are unstable to the initialization seeds [10] impacting human interpretability.

Idea: A neural network’s stability is connected to its Lipschitz constant [20]. Lipschitz-constrained neural networks augment stability by bounding its Lipschitz constant [18]. Exploring Lipschitz-constrained GNN explainers represents a promising research avenue.

REFERENCES

- [1] C. Abrate and F. Bonchi. Counterfactual graphs for explainable classification of brain networks. In *KDD*, page 2495–2504, 2021.
- [2] C. Abrate and F. Bonchi. Counterfactual graphs for explainable classification of brain networks. In *KDD*, 2021.
- [3] M. Bajaj, L. Chu, Z. Y. Xue, J. Pei, L. Wang, P. C.-H. Lam, and Y. Zhang. Robust counterfactual explanations on graph neural networks. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.
- [4] M. Bajaj, L. Chu, Z. Y. Xue, J. Pei, L. Wang, P. C.-H. Lam, and Y. Zhang. Robust counterfactual explanations on graph neural networks. *arXiv preprint arXiv:2107.04086*, 2021.
- [5] A. Derrow-Pinion, J. She, D. Wong, O. Lange, T. Hester, L. Perez, M. Nunkesser, S. Lee, X. Guo, B. Wiltshire, P. W. Battaglia, V. Gupta, A. Li, Z. Xu, A. Sanchez-Gonzalez, Y. Li, and P. Velickovic. Eta prediction with graph neural networks in google maps. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*. ACM, Oct. 2021.
- [6] N. Goyal, H. V. Jain, and S. Ranu. Graphgen: a scalable approach to domain-agnostic labeled graph generation. In *Proceedings of The Web Conference 2020*, pages 1253–1263, 2020.
- [7] Q. Huang, M. Yamada, Y. Tian, D. Singh, and Y. Chang. Graphlime: Local interpretable model explanations for graph neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 2022.
- [8] Z. Huang, M. Kosan, S. Medya, S. Ranu, and A. Singh. Global counterfactual explainer for graph neural networks. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*, pages 141–149, 2023.
- [9] M. Kosan, A. Silva, and A. Singh. Robust ante-hoc graph explainer using bilevel optimization, 2023.
- [10] M. Kosan, S. Verma, B. Armgaan, K. Pahwa, A. Singh, S. Medya, and S. Ranu. Gnnx-bench: Unravelling the utility of perturbation-based gnn explainers through in-depth benchmarking, 2023.
- [11] R. Lam, A. Sanchez-Gonzalez, M. Willson, P. Wirsberger, M. Fortunato, F. Alet, S. Ravuri, T. Ewalds, Z. Eaton-Rosen, W. Hu, A. Merose, S. Hoyer, G. Holland, O. Vinyals, J. Stott, A. Pritzel, S. Mohamed, and P. Battaglia. Learning skillful medium-range global weather forecasting. *Science*, 0(0):eadi2336, 2023.
- [12] W. Lin, H. Lan, and B. Li. Generative causal explanations for graph neural networks. In *International Conference on Machine Learning*, pages 6666–6679. PMLR, 2021.
- [13] A. Lucic, M. A. Ter Hoeve, G. Tolomei, M. De Rijke, and F. Silvestri. Cf-gnnexplainer: Counterfactual explanations for graph neural networks. In *AISTATS*, pages 4499–4511, 2022.
- [14] D. Luo, W. Cheng, D. Xu, W. Yu, B. Zong, H. Chen, and X. Zhang. Parameterized explainer for graph neural network. *Advances in neural information processing systems*, 33:19620–19631, 2020.
- [15] D. Luo, W. Cheng, D. Xu, W. Yu, B. Zong, H. Chen, and X. Zhang. Parameterized explainer for graph neural network. *arXiv preprint arXiv:2011.04573*, 2020.
- [16] A. Merchant, S. Batzner, S. Schoenholz, M. Aykol, G. Cheon, and E. Cubuk. Scaling deep learning for materials discovery. *Nature*, 624:1–6, 11 2023.
- [17] L. Rampásek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, and D. Beaini. Recipe for a General, Powerful, Scalable Graph Transformer. *Advances in Neural Information Processing Systems*, 35, 2022.
- [18] K. Scaman and A. Virmaux. Lipschitz regularity of deep neural networks: Analysis and efficient estimation. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS’18, page 3839–3848, 2018.
- [19] C. Shan, Y. Shen, Y. Zhang, X. Li, and D. Li. Reinforcement learning enhanced explainer for graph neural networks. In *NeurIPS 2021*, December 2021.
- [20] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.
- [21] J. Tan, S. Geng, Z. Fu, Y. Ge, S. Xu, Y. Li, and Y. Zhang. Learning and evaluating graph neural network explanations based on counterfactual and factual reasoning. In *Proceedings of the ACM Web Conference 2022*, WWW ’22, page 1018–1027, 2022.
- [22] J. Tan, S. Geng, Z. Fu, Y. Ge, S. Xu, Y. Li, and Y. Zhang. Learning and evaluating graph neural network explanations based on counterfactual and factual reasoning. In *WebConf*, 2022.
- [23] C. Vignac, I. Krawczuk, A. Siraudin, B. Wang, V. Cevher, and P. Frossard. Digress: Discrete denoising diffusion for graph generation. In *The Eleventh International Conference on Learning Representations*, 2023.
- [24] M. Vu and M. T. Thai. Pgm-explainer: Probabilistic graphical model explanations for graph neural networks. *Advances in neural information processing systems*, 33:12225–12235, 2020.
- [25] G. P. Wellawatte, A. Seshadri, and A. D. White. Model agnostic generation of counterfactual explanations for molecules. *Chemical science*, 13(13):3697–3705, 2022.
- [26] H. Xuanyuan, P. Barbiero, D. Georgiev, L. C.

- Magister, and P. Lió. Global concept-based interpretability for graph neural networks via neuron analysis. 2023.
- [27] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu. Do transformers really perform badly for graph representation? In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.
- [28] R. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32:9240, 2019.
- [29] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, page 974–983, 2018.
- [30] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32, 2019.
- [31] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *ICML, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 5694–5703. PMLR, 2018.
- [32] H. Yuan, J. Tang, X. Hu, and S. Ji. Xgnn: Towards model-level explanations of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 430–438, 2020.
- [33] H. Yuan, H. Yu, S. Gui, and S. Ji. Explainability in graph neural networks: A taxonomic survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.
- [34] H. Yuan, H. Yu, J. Wang, K. Li, and S. Ji. On explainability of graph neural networks via subgraph explorations. In *ICML*, pages 12241–12252. PMLR, 2021.

Report on the Second International Workshop on Data Systems Education (DataEd '23)

Daphne Miedema
Eindhoven University of Technology
Eindhoven, the Netherlands
d.e.miedema@tue.nl

Sihem Amer-Yahia
CNRS, Univ. Grenoble Alpes
Grenoble, France
ameryahs@univ-grenoble-alpes.fr

Michael Mior
Rochester Institute of Technology
Rochester, United States of America
mjmvc@rit.edu

Efthimia Aivaloglou
Delft University of Technology
Delft, the Netherlands
e.aivaloglou@tudelft.nl

George Fletcher
Eindhoven University of Technology
Eindhoven, the Netherlands
g.h.l.fletcher@tue.nl

Toni Taipalus
Tampere University
Tampere, Finland
toni.taipalus@tuni.fi

ABSTRACT

This report summarizes the outcomes of the second international workshop on Data Systems Education: Bridging Education Practice with Education Research (DataEd '23). The workshop was held in conjunction with the SIGMOD '23 conference in Seattle, USA on June 23, 2023. The aim of the workshop was to provide a dedicated venue for presenting and discussing data management systems education experiences and research by bringing together the database and the computing education research communities to share findings, to cross-pollinate perspectives and methods, and to shed light on opportunities for mutual progress in data systems education. The program featured two keynote talks, eight research paper presentations, and a discussion session. In this report, we present the workshop's main results, observations, and emerging research directions.

1. INTRODUCTION

Data systems education is foundational in programs such as computer science, data science, and information systems. The DataEd workshop¹ is organized as a dedicated venue for the presentation and discussion of data systems education research. DataEd took place for the first time at SIGMOD 2022 and provided the opportunity to discuss a broad range of topics, including data systems course and curriculum design, learning instruments, tools, and practices, ethics and responsibility, formative and summative assessment, and industry perspectives on data management knowledge and skills [1].

¹<https://dataedinitiative.github.io/>

DataEd focuses on the broad area of data systems education: the teaching and learning of databases, data management, and data systems topics, ranging across the whole field, from classical topics, such as physical design, query optimization, data modeling, data integration, visual analytics, and query languages to contemporary topics, such as machine learning for data management systems, data management for machine learning, large data science applications and pipelines, and responsible data management.

DataEd'23 took place at SIGMOD 2023 as a second iteration with a full-day workshop consisting of:

1. A keynote talk *Human Learners of Relational Query Processing: Who Cares?* by Sourav Bhowmick (Nanyang Technological University)
2. A keynote talk *SQL: A Trojan Horse Hiding a Decathlon of Complexities* by Toni Taipalus (University of Jyväskylä)
3. Eight research and tool paper presentations with accompanying discussions (50% acceptance rate)

In the following section, we present the themes that emerged from the various workshop activities.

2. WORKSHOP THEMES

2.1 Difficulties in Learning Query Languages

A common theme in Data Systems Education research is the analysis of student errors. As such, errors and difficulties were also a prominent theme in the second iteration of DataEd.

In *Human Learners of Relational Query Processing: Who Cares?* [10], Sourav S. Bhowmick discussed how query optimization is a challenging task to learn let alone

master, and novices struggle with learning DBMS internals. Challenges include understanding query execution plans, incorrect ordering of steps, missing intermediate relations, and understanding cost estimation. What's more, DBMS vendors primarily target enterprise users for SQL training due to financial incentives, consequently disregarding novices learning query processing and optimization. Off-the-shelf DBMSs simply do not offer enough feedback on their query execution plans, possibly making learning query processing too challenging for effective learning, given learners of different abilities and backgrounds. To assist in learning query processing through more helpful, natural language feedback and query visualization tools such as ARENA [14], LANTERN [4] and MOCHA [13] were suggested.

In *SQL: A Trojan Horse Hiding a Decathlon of Complexities* [12], Toni Taipalus argued for acknowledging the gap between how simple SQL *looks* and how complicated the language actually *is* by highlighting ten complexities arising from the discrepancies between SQL's underlying principles, the language as it is defined by the SQL Standard, and implementations of SQL in various DBMSs. By acknowledging convolutions such as conflicts between the theory of three-valued logic and how three-valued logic is implemented, confusing error messages, and strange conventions regarding grouping, novices can better brace themselves for learning a challenging language from the get-go. Similarly to Bhowmick, Taipalus suggested using query visualization tools, and additionally focusing on teaching one SQL dialect with a DBMS that closely conforms to the SQL Standard, and refraining from treating SQL like a natural language.

Some of these concerns are further highlighted in *Student's Learning Challenges with Relational, Document, and Graph Query Languages* presented by Abdussalam Alawini [2]. The authors highlighted several challenges in learning query languages faced by students by analyzing over 350,000 student submissions. The authors found that semantic errors (incorrect results) in SQL queries were indeed a problem, with 35% of student submissions experiencing some kind of logic error. This problem was even more prevalent with MongoDB queries, where two-thirds of submissions contained a semantic error. Cypher queries on Neo4j also exhibited a significant error rate, with 40% of submissions containing a semantic error. The authors posit that students may have a more difficult time forming a mental model for document databases compared to relational and graph databases. This suggests a need to further explore methods for teaching query languages beyond SQL, particularly for NoSQL databases.

As one possible approach to furthering query language education, Michael Mior proposed *Relational Playground*:

Teaching the Duality of Relational Algebra and SQL [8]. Relational Playground aims to provide students with the ability to thoroughly explore the connection between relational algebra and SQL. Students can enter SQL queries against a sample database and view the corresponding relational algebra expressions. The interface also makes it possible to view intermediate results at each stage of execution as well as the effects of some basic query optimization techniques. Further evaluation of the tool is required, but it shows promise at cementing student understanding of SQL query processing.

2.2 Tools/Automating Assessment

A second major theme is that of tool development. In this iteration of DataEd, we had four papers on tooling for education, to support students, teachers, or both.

The first talk was by Daniel Kocher, who discussed *Feedforward-Aided Course Designs for Similarity* [5]. The course designs he presented are called project-based learning and task-based learning and are of use for other teachers to reflect on and potentially adapt. In both course designs, they employ an auto-grader to provide students with automated and instant feedforward. This helps both students and lecturers, as it removes the workload of grading. It also supported students working with different programming languages, creating even more flexibility. However, they found that the heterogeneity of students also led to struggles, such as in different levels of programming and conceptualization knowledge. The discussion afterward led to suggestions for future development, such as creating individual feedforward and reducing the opportunities for gaming the system.

The second talk in this category was by Siheem Amer-Yahia, titled *Adaptive Test Recommendation for Mastery Learning* [3], where the authors presented several solutions for adaptive educational systems. First, the authors formalized the Adaptive Upskilling Problem (ADUP) as a multi-objective optimization problem to select a batch of tests that maximize expected performance and aptitude while minimizing the skill gap for learners. Second, a heuristic algorithm based on Hill Climbing was developed to find a subset of Pareto solutions for the ADUP Problem. This algorithm helps in selecting the most appropriate tests for learners to improve their skills efficiently. Finally, the authors used simulations to compare different variants of the problem and confirmed the effectiveness of the proposed approach in achieving skill mastery [9].

Next, Ruben Mayer discussed *pTA: An Automated Teaching Assistant for Lab Courses* [11]. Lab courses allow for active learning, which is more effective than learning passively from lectures. However, the workload associated with this is much higher too. At the Technical University of Munich, they developed pTA,

which can evaluate students' solutions to a Cloud Database course, allowing for instant feedback to the students. Previously, the course was at maximum capacity, but pTA created space for more students due to the lowered teaching load. It also helped by generating a grade report for the submissions and allowed students to work more independently by studying the logs. In the later discussion, Ruben elaborated on some of the avenues for future work, such as connecting to other universities to test the platform and improving error messages within the system.

Finally, Sophia Yang discussed work on *Mining SQL Problem Solving Patterns using Advanced Sequence Processing Algorithms* [15]. This is a follow-up work on their DataEd'22 paper, where they found that global analysis was too time-consuming. Therefore, they decided to re-encode students' problem-solving patterns (gathered from solution scores) employing symbols. They then performed sequence compaction to shorten repeated equal patterns. They found that some patterns represent code testing, and others represent incorrect thought patterns. The encoding also allows one to easily extract questions with a single answer, which form suspicious behavior. Overall, the encoding of student behavior with symbols can easily be adopted by other teachers to make students' learning progress more actionable.

Overall, this iteration of DataEd provided a new set of tools to support teaching and assessment.

2.3 DataEd for Non-Computing Students

The last major theme concerned non-computing students; students majoring in something other than computer science or computer engineering. This student population is increasingly interested in data science, as it can provide them with tools to work more effectively with data. For teachers, it is important to distinguish between majoring and non-majoring students, as their context is different. As such, teachers likely need to approach these students differently.

The first talk highlighting this was presented by Erik Golen, titled *Offering Data Science Education to Non-Computing Majors* [7]. The central theme in this work is how traditional (data science) courses are not suitable for non-computing majors. Erik and his colleagues suggest taking a hands-on, low-code approach. However, they also find that, for low-code courses, some problems in course design include the definition of achieving desirable learning outcomes, solving problems in varying domains, and creating accessible lab environments. The course that they designed to work around these problems involved using datasets from the students' own domain, such as film, communication, political science, and history. The full course design is described in the paper and was well received by the students.

A set of web-based visualization tools was presented by Sean Kross for the paper *Teaching Data Science by Visualizing Data Table Transformations: Pandas Tutor for Python, Tidy Data Tutor for R, and SQL Tutor* [6]. The work is motivated by the observation that enrollments in introductory data classes in many universities have increased due to interest from across disciplines and that it is hard for students to understand individual code statements and the semantic differences between data languages. The presented set of tools² support this for introductory data courses by rendering step-by-step diagrams, from input to output, of data table transformations such as filtering, sorting, reshaping, pivoting, grouping, and joining, expressed in Python, R, and SQL. Behind the tools is a table visualization library that illustrates the relationships between rows, columns, and cells of operations' input and output tables. At the time of the workshop, deployment was limited, and so we cannot report on students' experiences, but instructor interest has been high.

Discussion within this topic moved to the influence of programming experience on performance. Anecdotally for the DataEd community, students without experience seem to regularly perform better. Some attendees speculate that this might be due to students with experience being complacent, and students without the experience being aware they start off 'behind'.

3. CLOSING DISCUSSION AND EMERGING RESEARCH DIRECTIONS

DataEd concluded with a discussion of recurring topics gathered throughout the day. The topics and open questions included:

- There is a wealth of theory in psychology and education that we can borrow from and build better educational materials and tools with.
- How do we teach some of the following topics: conceptual modeling, (de)normalization, NoSQL approaches, embedded SQL, and advanced SQL keywords such as JOIN and GROUP BY.
- What topics belong in an introductory data systems course?
- How do tools like Spark and Flink fit into Data Systems Education?
- What is the role of LLMs in Data Systems Education?

These topics provide new challenges and research directions for the field of Data Systems Education.

²<https://tidydatatutor.com/>, <https://pandastutor.com/>, and <https://cudbg.github.io/sqltutor/>

4. CONCLUSIONS AND IMPLICATIONS

DataEd'23 was another highly successful event, with a good number of submissions, interesting talks, and high attendance. This document describes some of the most pressing topics in Data Systems Education, which researchers in the area can use to further the field.

The research and findings from this second edition of DataEd have various implications, both for industry and educators. For *industry*, the usability of off-the-shelf products could be improved by increasing the level of feedback available to users on inner workings. Some examples include the explanation of query plans and the development of more accessible error messages. For *educators*, the increase of interest in data science from students of different backgrounds leads to challenges in teaching. These students have different backgrounds, and as such may benefit from a different approach to teaching, with increased scaffolding and the use of more engaging data. The tools discussed in subsection 2.2 may help educators by reducing the teaching load.

5. REFERENCES

- [1] Efthimia Aivaloglou, George Fletcher, Michael Liut, and Daphne Miedema. Report on the First International Workshop on Data Systems Education (DataEd'22). *ACM SIGMOD Record*, 51(4):49–53, 2023.
- [2] Ridha Alkhabaz, Zepei Li, Sophia Yang, and Abdussalam Alawini. Student's learning challenges with relational, document, and graph query languages. In *Proceedings of the 2nd International Workshop on Data Systems Education: Bridging Education Practice with Education Research*, DataEd '23, page 30–36, New York, NY, USA, 2023. Association for Computing Machinery.
- [3] Nassim Bouarour, Idir Benouaret, Cédric D'Ham, and Sihem Amer-Yahia. Adaptive test recommendation for mastery learning. In *Proceedings of the 2nd International Workshop on Data Systems Education: Bridging Education Practice with Education Research*, DataEd '23, page 18–23, New York, NY, USA, 2023. Association for Computing Machinery.
- [4] Peng Chen, Hui Li, Sourav S. Bhowmick, Shafiq R. Joty, and Weiguo Wang. LANTERN: Boredom-conscious natural language description generation of query execution plans for database education. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 2413–2416, New York, NY, USA, 2022. Association for Computing Machinery.
- [5] Thomas Hütter and Daniel Kocher. Feedforward-aided course designs for similarity search. In *Proceedings of the 2nd International Workshop on Data Systems Education: Bridging Education Practice with Education Research*, DataEd '23, page 14–17, New York, NY, USA, 2023. Association for Computing Machinery.
- [6] Sam Lau, Sean Kross, Eugene Wu, and Philip J. Guo. Teaching data science by visualizing data table transformations: Pandas tutor for Python, tidy data tutor for R, and SQL Tutor. In *Proceedings of the 2nd International Workshop on Data Systems Education: Bridging Education Practice with Education Research*, DataEd '23, page 50–55, New York, NY, USA, 2023. Association for Computing Machinery.
- [7] Xumin Liu, Erik Golen, Rajendra K. Raj, and Kimberly Fluet. Offering data science coursework to non-computing majors. In *Proceedings of the 2nd International Workshop on Data Systems Education: Bridging Education Practice with Education Research*, DataEd '23, page 44–49, New York, NY, USA, 2023. Association for Computing Machinery.
- [8] Michael J. Mior. Relational playground: Teaching the duality of relational algebra and SQL. In *Proceedings of the 2nd International Workshop on Data Systems Education: Bridging Education Practice with Education Research*, DataEd '23, page 56–58, New York, NY, USA, 2023. Association for Computing Machinery.
- [9] Radek Pelánek and Jiří Říhák. Experimental analysis of mastery learning criteria. In *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization*, pages 156–163, 2017.
- [10] Sourav S Bhowmick. Human learners of relational query processing: Who cares? In *Proceedings of the 2nd International Workshop on Data Systems Education: Bridging Education Practice with Education Research*, DataEd '23, page 1–8, New York, NY, USA, 2023. Association for Computing Machinery.
- [11] Jawad Tahir, Raj Mandal, Olha Stefanova, Hans-Arno Jacobsen, Christoph Doblander, and Ruben Mayer. pTA: A programmable teaching assistant for lab courses. In *Proceedings of the 2nd International Workshop on Data Systems Education: Bridging Education Practice with Education Research*, DataEd '23, page 24–29, New York, NY, USA, 2023. Association for Computing Machinery.
- [12] Toni Taipalus. SQL: A Trojan horse hiding a decathlon of complexities. In *Proceedings of the*

2nd International Workshop on Data Systems Education: Bridging Education Practice with Education Research, DataEd '23, page 9–13, New York, NY, USA, 2023. Association for Computing Machinery.

- [13] Jess Tan, Desmond Yeo, Rachael Neoh, Huey-Eng Chua, and Sourav S Bhowmick. MOCHA: A tool for visualizing impact of operator choices in query execution plans for database education. *Proc. VLDB Endow.*, 15(12):3602–3605, aug 2022.
- [14] Hu Wang, Hui Li, Sourav S Bhowmick, and Baochao Xu. ARENA: Alternative relational query plan exploration for database education. *SIGMOD '23*, page 107–110, New York, NY, USA, 2023. Association for Computing Machinery.
- [15] Sophia Yang, Geoffrey L. Herman, and Abdussalam Alawini. Mining SQL problem solving patterns using advanced sequence processing algorithms. In *Proceedings of the 2nd International Workshop on Data Systems Education: Bridging Education Practice with Education Research*, DataEd '23, page 37–43, New York, NY, USA, 2023. Association for Computing Machinery.