SIGMOD Officers, Committees, and Awardees

Chair	Vice-Chair	Secretary/Treasurer
Divyakant Agrawal	Fatma Ozcan	Rachel Pottinger
Department of Computer Science	Systems Research Group	Department of Computer Science
UC Santa Barbara	Google	University of British Columbia
Santa Barbara, California	Sunnyvale, California	Vancouver
USA	USA	Canada
+1 805 893 4385	+1 669 264 9238	+1 604 822 0436
agrawal <at> cs.ucsb.edu	Fozcan <at> google.com	Rap <at> cs.ubc.ca

Jim Gray Doctoral Dissertation Award Committee:

Wolfgang Lehner (co-chair), Gustavo Alonso (co-chair), Azza Abouzied, Daniel Deutch, Evaggelia Pitoura, Xiaofang Zhou, Huanchen Zhang, and Chenggang Wu

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)	

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 Nandrekar-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 Fora, 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)

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 Özsoyoğlu (2018)
Ahmed Elmagarmid (2019)	Philipe 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)		

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** *Winne*r: Peter Bailis. *Honorable Mention*: Immanuel Trummer
- **2018** *Winne*r: Viktor Leis. *Honorable Mention*: Luis Galárraga and Yongjoo Park
- **2019** *Winne*r: 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

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

[Last updated: June 1, 2023]

Editor’s Notes

Welcome to the December 2023 issue of the ACM SIGMOD Record!

This issue starts with the Database Principles column presenting an article by ten Cate and colleagues on algorithms for solving the fitting problem for conjunctive queries. The fitting problem, which concerns constructing queries that fit the given labeled data examples, has a long history in database research, with connections ranging from the query-by-example paradigm to inductive logic programming. The authors consider the fitting problem for the case of conjunctive queries, that is select-project-join queries with equality comparisons, covering desirable properties for potential algorithms addressing the problem, a comparison of existing algorithms under these considerations, and structural interactions between the considerations. The article concludes with an outline of interesting directions for further investigation.

The Surveys column features a contribution by Siddiqui and Wu. The article considers the landscape of challenges in recommending high-quality indexes while ensuring scalability in managing workloads in modern cloud services. Within this scope, the authors explore ways in which machine-learning techniques provide new opportunities in the mitigation of some of the challenges in automated index tuning. The article highlights the key takeaways from the recent efforts in these directions, and underlines the gaps that need to be closed for their effective functioning within the traditional index-tuning framework. The information provided in the article can be useful in providing context and impetus to the further research and development efforts in automated index tuning.

The Reminiscences on Influential Papers column features contributions by Sourav Bhowmick, Carsten Binnig, and Peter Alvaro.

The Advice to Mid-Career Researchers column presents a contribution by Tiziana Catarci. In the column, she shares her experiences with the university-career tracks in her home country and in Europe as a whole, and provides perspective on how the mid-career period can give researchers even more freedom to study, create their own groups, and choose their own ventures. She also explores potential ways to impact the society, to contribute to the creation of a better world, and to be a positive example for the younger generation during that career period.

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 Tianzheng Wang. The article ponders research questions around the roles of programming languages in implementing DBMS. The DBMS software is expected to deliver both on good use of the hardware for high performance and on success with high-level abstractions, which often leads to the choice of more than one programming language for the implementations. The article makes a case for exploring features of programming languages in deciding whether a single programming language can be used in implementing DBMS, and for quantitatively comparing the available programming languages for this purpose. As compiler and ecosystem support can often fall short in DBMS development, the author calls on the community to consider influencing programming-language and compiler design to push the desired features early into future programming languages.

The issue closes with an Open Forum column, which presents an article by Meliou and colleagues that focuses on the important and hard task of reviewing papers for conferences. The article reports the results of a survey conducted to gather the opinions of the data-management community regarding what could be done to address the existing challenges in conference reviewing. The authors reached out to about 1,200 members of the community with relevant reviewing experience, and collected 345 anonymous responses. A planned follow-up to the article will discuss in more depth particular proposals inspired by the collective feedback from the community.

On behalf of the SIGMOD Record Editorial board, I hope that you enjoy reading the December 2023 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

December 2023

Fitting Algorithms for Conjunctive Queries

Balder ten Cate
ILLC, University of Amsterdam

Maurice Funk
Leipzig University and ScaDS.AI

Jean Christoph Jung
TU Dortmund University

Carsten Lutz
Leipzig University and ScaDS.AI

ABSTRACT

A fitting algorithm for conjunctive queries (CQs) is an algorithm that takes as input a collection of data examples and outputs a CQ that fits the examples. In this column, we propose a set of desirable properties of such algorithms and use this as a guide for surveying results from the authors' recent papers published in PODS 2023, IJCAI 2023, and *Inf. Proc. Letters* 2024. In particular, we explain and compare several concrete fitting algorithms, and we discuss complexity and size bounds for constructing fitting CQs with desirable properties.

1. INTRODUCTION

The *fitting problem* for conjunctive queries (CQs) is the problem to construct a CQ q that fits a given set of labeled data examples. This fundamental problem has a long history in database research. It lies at the heart of the classic *Query-By-Example* paradigm [41] that aims to assist users in query formation and query refinement, and that is also known as *query reverse engineering*. It has been intensively studied for CQs [38, 33, 7] and other types of queries (e.g., [10, 4, 20]). The fitting problem is also central to *Inductive Logic Programming* [21, 27], where CQs correspond to the basic case of non-recursive single-rule Datalog programs, and has close connections to fitting problems for *schema mappings* [2, 12]. More recent motivation comes from *automatic feature generation in machine learning with relational data* [30, 8]. Here, the CQ fitting problem arises because a CQ that separates positive from negative examples in (a sufficiently large subset of) a labeled dataset is a natural contender for being added as an input feature to the model [8].

Examples illustrating the fitting problem are given in Table 1. In each example, certain values from the database instance are labeled as positive and/or negative, and a fitting query must include the positive examples in its output and exclude the negative examples.

Depending on the application, desirable properties of a fitting algorithm may include the following:

[1]This paper was written in response to a dual invitation from the SIGMOD Record Database Principles column and the ACM SIGMOD Research Highlight Award committee.

Efficiency Ideally, the fitting algorithm should run in polynomial time in the size of the input examples, or at least in polynomial time in the size of the input examples plus the size of the smallest fitting query. We call the latter *weakly polynomial.*

Succinctness Ideally, the fitting algorithm outputs a query of small size, i.e., not much larger than the smallest fitting query. This can be formalized through the notion of the "Occam property", which requires that the size of the output concept is bounded polynomially in the size of a target query and sublinearly in the number of input examples.

Producing Extremal Fittings When several fitting queries exist, it may be desirable to output a query that is either most-general or most-specific among all fitting queries.

Generalization to Unseen Examples Ideally, we would like the fitting algorithm to come with a (probabilistic) guarantee that the output query not only fits the input examples, but performs well on future unseen examples (drawn from the same distribution as the input examples and labeled according to the same "target query"). This is formalized by the well-known PAC ("Probably Approximately Correct") property.

Completeness for Design Ideally, the fitting algorithm can be compelled to produce any CQ (up to equivalence) by giving it the right input examples.

It turns out that some of these properties are difficult or impossible to attain. Also, there are properties that can be obtained in isolation, but not in combination. A fundamental difficulty is that the fitting problem is computationally hard, and it is known that the smallest fitting CQ for a given collection of labeled examples is in general exponential in size [40]. To make things worse, the hardness pertains already to very small subclasses of CQs and natural restrictions on the fitting problem obtained by requiring, for instance, that all input examples use the same database instance (cf. Section 4).

Example database instance I:

Businessman
donald
fred
james

Democrat
barack
franklin

Father	
barack-sr	barack
fred	donald
james	franklin

Republican
donald

Economist
barack-sr

Labeled examples:	A fitting query:
$E^+ = \{(I, \text{franklin}), (I, \text{barack})\}$ $E^- = \{(I, \text{donald})\}$	$q(x)$:- Democrat(x).
$E^+ = \{(I, \text{franklin}), (I, \text{donald})\}$ $E^- = \{(I, \text{barack})\}$	$q(x)$:- Father(y, x), Businessman(y).
$E^+ = \{(I, \text{barack}), (I, \text{donald})\}$ $E^- = \{(I, \text{franklin})\}$	the union of CQs $q(x)$:- Republican(x) $q(x)$:- Father(y, x), Economist(y).

Table 1: Examples of the fitting problem

$$\begin{tikzcd}
\text{Strongly Occam} \arrow[r, Rightarrow] \arrow[d, Rightarrow] & \text{Occam} \arrow[r, Rightarrow] & \text{Polynomial PAC} \arrow[dl, dash, "\times"] \arrow[d, dash, "\times"] \\
\text{Complete for Design} & \text{Producing a most-specific fitting} \arrow[l, Rightarrow] \arrow[r, dash, "\times"] & \text{Producing a most-general fitting when it exists}
\end{tikzcd}$$

Figure 1: Summary of interactions between desirable properties (the edge label $\times$ means incompatibility)

Despite these roadblocks, there are ways forward. One may opt to bite the bullet and live with the high computational complexity, for example by employing SAT solvers. One may also extend the capabilities of the fitting algorithm by allowing it to interactively ask questions to an oracle that provides black-box access to a fitting CQ (the "target CQ"). Specifically, the algorithm may produce a database instance and a tuple and ask whether the tuple belongs to the output of the target CQ on that instance. In reality, the oracle may be a preexisting compiled CQ (in reverse-engineering scenarios) or a user (in interactive query specification settings). A third option is to resort to incomplete approaches that do not always return a fitting CQ when one exists.

In this column, we (i) compare several different fitting algorithms based on the above considerations and (ii) discuss structural interactions between the above desiderata. In particular, we will study three fitting algorithms for CQs, described at a high level as follows:

Algorithm P This algorithm takes as input a collection of labeled examples and outputs the canonical CQ of the direct product of the positive examples, which is known to fit whenever a fitting CQ exists.

Algorithm M can be viewed as an optimized version of Algorithm P that additionally interacts with an oracle as described above. It computes the product of the positive examples in an iterative way and uses the oracle to minimize the query in each iteration.

Algorithm B proceeds by solving a size-bounded version of the fitting problem for increasing size bounds, until a fitting CQ is found. This is also known as the "bounded fitting approach".

Each of the five desirable properties listed above will be discussed in a separate section of this column. We will discuss how the three algorithms fare, explain how this is complemented by lower bounds, and discuss structural interactions between the different desiderata. Table 2 provides a summary of the interactions between the considered properties. The technical results we present are mostly drawn from the recent papers [14, 17, 16]. Indeed, our motivation for writing this column is to fit these results into a unified picture.

Fitting problems for CQs and especially also for broader classes of logic programs have been investigated extensively in the literature on *Inductive Logic Programming (ILP)*, with a strong emphasis on systems implementation. The most common technique used in ILP systems is based on *refinement operators*. In the case of CQs it is known that fitting algorithms based on refinement operators are incomplete. Nevertheless, refinement-operator based approaches have been successfully implemented and used. Therefore, in Section 9, we also discuss them and their relationship to our algorithms.

Outline. In Section 2, we define the fitting problem. In Section 3 we describe the three algorithms in more detail. In Sections 4–8, we investigate the above five desiderata one by one. In Section 9, we discuss refinement-based approaches to fitting. Finally, we conclude in Section 10.

2. THE FITTING PROBLEM FOR CQS

As usual, a schema $\mathcal{S}$ is a set of relation symbols, each with associated arity. A *database instance* over $\mathcal{S}$ is a finite set I of *facts* of the form $R(a_1, \ldots, a_n)$ where $R \in \mathcal{S}$ is a relation symbol of arity n and $a_1, \ldots, a_n$ are *values*. We use $adom(I)$ to denote the set of all values used in I. We can then view a *query* over a schema $\mathcal{S}$, semantically, as a function q that maps each database instance I over $\mathcal{S}$ to a set of k-tuples $q(I) \subseteq adom(I)^k$, where $k \geq 0$ is the *arity* of the query.

Data Examples. A *data example* for a query q consists of a database instance I together with information about the intended query output $q(I)$. We focus on two types of data examples (cf. Remark 2.4 for other types):

- a *positive example* for q is a pair $(I, \mathbf{a})$ with $\mathbf{a} \in q(I)$;
- a *negative example* for q is a pair $(I, \mathbf{a})$ with $\mathbf{a} \in adom(I)^k \setminus q(I)$

	Needs oracle	Running time	Extremal fitting	Occam	PAC	Complete for design
Alg P	no	Exponential in the input (*not* weakly polynomial)	yes (most-specific)	no	no	yes
Alg M	yes	Exponential in the input (weakly polynomial)	no	yes (strongly)	yes (polynomially)	yes
Alg B	no	Doubly exponential (*not* weakly polynomial)	no	yes (strongly)	yes (polynomially)	yes

Table 2: Summary of our comparison of fitting algorithms

with k the arity of the query q. We also say that k is the arity of the example.

By a *collection of labeled examples* we mean a pair $E = (E^+, E^-)$, where E^+ and E^- are sets of examples. We say that a query q *fits* E if each member of E^+ is a positive example for q and each member of E^- is a negative example for q, or, in other words, if $\mathbf{a} \in q(I)$ for all $(I, \mathbf{a}) \in E^+$ and $\mathbf{a} \notin q(I)$ for all $(I, \mathbf{a}) \in E^-$. Note that for a fitting to exist, all examples in the collection must have the same arity.

Example 2.1. *Consider the database instance I depicted in Table 1, over a schema that consists of unary and binary relation symbols, storing information about American presidents. Table 1 lists several collections of labeled examples, and, for each, a fitting query. Note that in the third case, no fitting conjunctive query exists, but a fitting union of conjunctive queries exists.*

Fitting Problems. We discuss three algorithmic problems related to fitting, relative to a query language $\mathcal{Q}$:

Fitting Construction Given a collection E of labeled examples that has a fitting query in $\mathcal{Q}$, construct such a query.

Fitting Existence Given a collection E of labeled examples, decide whether a fitting query from $\mathcal{Q}$ exists.

Fitting Verification Given a collection E of labeled examples and a query $q \in \mathcal{Q}$, decide if q fits E.

Among the three problems above, fitting construction is the main problem of interest and we sometimes refer to it also as the *fitting problem.* Note that we have formulated fitting construction as a promise problem, in order to study its complexity in isolation from the fitting existence problem. In practice, one may combine a fitting construction algorithm with a fitting existence test.

A solution to the fitting construction problem, as defined above, is *any* query that fits. In particular, the query is not required to generalize from the input collection of labeled examples to other examples, i.e., there is no penalty for overfitting.

If $\mathcal{Q}$ is the class of all relational algebra queries, the above problems are trivial. Indeed, if constants are admitted in the query, a fitting relational algebra query exists for (E^+, E^-) if and only if $E^+ \cap E^- = \emptyset$; as a fitting query one can pick, intuitively, the union of the complete descriptions of the positive examples. Without constants, a fitting relational algebra query exists if and only if no member of E^+ is isomorphic to a member of E^- (cf. [23]). This situation changes for more restricted query languages $\mathcal{Q}$; we consider conjunctive queries.

Conjunctive Queries and tree CQs. By a k-ary *conjunctive query (CQ)* over a schema $\mathcal{S}$, we mean an expression of the form $q(\mathbf{x}) \text{ :- } \alpha_1, \ldots, \alpha_n$ where $\mathbf{x} = x_1, \ldots, x_k$ is a sequence of variables and each α_i is a relational atom that uses a relation symbol from $\mathcal{S}$ and no constants. The variables in $\mathbf{x}$ are called *answer variables* and the other variables used in the atoms α_i are the *existential variables.* Each answer variable is required to occur in at least one atom α_i, a requirement known as the *safety condition.* A CQ of arity 0 is called *Boolean.* With the *size* of a CQ, we mean the number of atoms in it. The query output $q(I)$ is defined as usual, cf. any standard database textbook. Two CQs q_1 and q_2 are *equivalent*, written $q_1 \equiv q_2$ if $q_1(I) = q_2(I)$ for all database instances I.

Besides CQs we will also consider a more restricted class of queries, namely *tree CQs.* In order to define it, we need to talk about canonical database instances. The canonical database instance of a CQ q is the instance I_q (over the same schema as q) whose active domain consists of the variables that occur in q and whose facts are the atomic formulas in q. The definition of tree CQs is restricted to schemas that consist of unary and binary relation symbols only. Note that every instance over such a schema can naturally be viewed as a directed, edge-labeled and node-labeled graph. A *tree CQ* is then a unary CQ $q(x)$ such that I_q, viewed in the above way, is a directed node-labeled and edge-labeled tree with root x (without parallel edges, and where all edges are directed away from the root). Our interest in this class of CQs stems from the fact that they form a notational variant of concept expressions in the description logic $\mathcal{EL}$, and that they are in some ways computationally more attractive.

The fitting construction, fitting existence, and fitting verification problems have been studied extensively for the case of CQs as well as for the case of tree CQs. We will review the known results in the subsequent sections. Here, we briefly comment on the fitting verification problem. It was observed in [14] that this problem is DP-complete for CQs, where DP is the class of problems that are the intersection of a problem in NP and a problem in coNP. Tree CQs, on the other hand, can be evaluated in polynomial time (combined complexity) and it follows that the fitting verification problem for tree CQs is solvable in polynomial time (cf. [14]). In summary:

THEOREM 2.2 ([14]). *Fitting verification is DP-complete for CQs and in PTime for tree CQs.*

REMARK 2.3. *As defined above, CQs do not contain constants, so we may not write, for instance, $q(x)$:- $R(x, 100)$. Filipetto [22] studied the impact of allowing constants on the fitting problem. For CQs, it turns out, whether we allow constants does not affect the fitting problem, modulo simple reductions. Specifically, given a collection of labeled examples E, we can take an isomorphic copy E' of E in which every constant is renamed, and take the union $E \cup E'$. Then (assuming E contains at least one positive example) the CQs that fit $E \cup E'$ are precisely the CQs that do not contain constants and fit E. Conversely, constants can be simulated by unary relations. The same applies to tree CQs.*

While we do not consider unions of conjunctive queries (UCQs) here, it is interesting to point out that they behave differently: allowing constants in UCQs trivializes the fitting problem. There are several ways to address this, including specifying a set of allowed constants as part of the input to the fitting problem, or restricting the number of allowed constants in the query. Both variants can be solved by a reduction to the constant-free fitting problem. Furthermore, identifying a small set of meaningful constants from data is an interesting problem by itself that deserves further study. See [22] for more details.

REMARK 2.4. *Depending on the application scenario, it may be natural to consider other types of examples besides positive and negative examples. In particular, an* input-output example *is a pair $(I, q(I))$ consisting of a database instance together with the entire query output. Such an example can be viewed as a succinct representation of a collection of $|adom(I)|^k$ many positive and negative examples, and therefore all the fitting algorithms that we will discuss can also be applied to input-output examples, although the complexity bounds do not necessarily carry over. See also [13, Section 6].*

Also, depending on the application scenario, it may be the case that E^+ and E^- consist of examples with the same database instance. Many of the complexity bounds we will discuss hold already in this restricted setting.

3. THREE FITTING ALGORITHMS

We now define in detail the three algorithms for the fitting construction problem for CQs.

Algorithm 3.1: Algorithm P

```
Input  : collection $(E^+, E^-)$ of labeled examples for
         which a fitting CQ exists
Output : a fitting CQ
1 $e^* := e^k_\top$ where $k$ is the arity of $(E^+, E^-)$;
2 foreach $e \in E^+$ do
3     $e^* := e^* \times e$
4 return the canonical CQ of $e^*$
```

Algorithm P. This algorithm simply returns the canonical CQ of the direct product of the positive examples. Intuitively, it extracts the commonalities of the positive examples. We make this more precise.

The *canonical CQ* of a data example $(I, a_1, \ldots, a_n)$ is simply the CQ $q(x_{a_1}, \ldots, x_{a_n})$ whose atoms are the facts of I, where each value $a \in adom(I)$ is uniformly replaced by a fresh variable x_a.

The *direct product* $I \times J$ of two instances I and J (over the same schema), is the database instance over $\mathcal{S}$ that consists of all facts $R(\langle a_1, b_1\rangle, \langle a_2, b_2\rangle, \ldots, \langle a_n, b_n\rangle)$, where $R(a_1, \ldots, a_n)$ is a fact in I and $R(b_1, \ldots, b_n)$ is a fact in J. Note that the active domain of $I \times J$ consists of pairs from $adom(I) \times adom(J)$. The direct product $(I, \mathbf{a}) \times (J, \mathbf{b})$ of two examples, with $\mathbf{a} = a_1, \ldots, a_k$ and $\mathbf{b} = b_1, \ldots, b_k$ of the same length, is given by $(I \times J, (\langle a_1, b_1\rangle, \langle a_2, b_2\rangle \ldots, \langle a_k, b_k\rangle))$. In general, this may not yield a well-defined example because there is no guarantee that the distinguished elements $\langle a_1, b_1\rangle, \langle a_2, b_2\rangle, \ldots, \langle a_k, b_k\rangle$ belong to $adom(I \times J)$. For a finite set of examples $E = \{e_1, \ldots, e_n\}$, we write $\prod_{e \in E}(e)$ for the direct product $e_1 \times \cdots \times e_n$ (note that $\times$ is associative up to isomorphism). The direct product operation should not be confused with the Cartesian product operation from relational algebra: the former preserves the schema, including the arity of each relation, but changes the domain; the latter preserves the domain but produces a relation of increased arity.

We need one more notion. With $e^k_\top$, we denote the strongest example of arity k. More precisely, $e^k_\top$ takes the form $(I, \mathbf{a})$ where I is an instance with a single value a that contains all possible facts over the schema and a, and $\mathbf{a}$ is the tuple $(a, \ldots, a)$ of length k. With these notions in place, Algorithm P is given as Algorithm 3.1. It simply computes $\Pi_{e \in E^+}(e)$, the negative examples are not used. Note that taking the product of n instances may result in an active domain of size exponential in n.

EXAMPLE 3.1. *In almost all of the remaining examples in this paper and without further notice, we use a schema $\mathcal{S}$ that consists of a single binary relation symbol R and consider Boolean queries. Note that, then, we may view an example simply as a database instance, without distinguished values. Also, many examples will refer to database instances that take the form of a cycle. For every $i \geq 1$, we use C_i to denote the database instance that*

Algorithm 3.2: Algorithm M

```
Input  : collection $(E^+, E^-)$ of labeled examples for
         which a fitting CQ exists
Output : a fitting CQ
1 $e^* := e_\top^k$ where $k$ is the arity of $(E^+, E^-)$;
2 foreach $e \in E^+$ do
3     $e^* := e^* \times e$;
4     $e^* :=$ minimize$(e^*)$;
5 return the canonical CQ of $e^*$
6 procedure minimize$(I, \mathbf{a})$
7 foreach $f \in I$ do
8     if $\mathbf{a} \in \widehat{q}(I \setminus \{f\})$ according to $MEMB_{\widehat{q}}$ then
9         $I := I \setminus \{f\}$
```

consists of the facts $R(a_1, a_2), R(a_2, a_3), \ldots, R(a_i, a_1)$. In other words, C_i is a cycle with i edges.

Now consider the collection of labeled examples $E = (E^+, E^-)$ where $E^+ = \{C_2, C_3\}$ and E^- consists of the single-fact instance $\{R(a, b)\}$. Then Algorithm P outputs the canonical CQ of C_6. Note that the direct product of any two instances C_n and C_m with n, m prime is isomorphic to $C_{n \cdot m}$. Clearly, this is a fitting CQ for E.

The following well-known fact (cf. for instance [40]) implies the correctness of Algorithm P.

THEOREM 3.2. *If any CQ fits a collection of labeled examples $E = (E^+, E^-)$, then the canonical CQ of the direct product $\Pi_{e \in E^+}(e)$ is well-defined and fits E.*

This also implies that Algorithm P can be turned into an algorithm for fitting existence by checking whether the constructed CQ fits the negative examples.

Algorithm M. We next consider Algorithm M, first given in [17]. Algorithm M differs from Algorithms P and B in having access via a membership oracle (whence the "M") to a concrete target CQ $\widehat{q}$ that fits the collection of examples given as an input. Given an example e, the oracle returns (in unit time) the status of e, that is, whether e is a positive or negative example for $\widehat{q}$. We denote such an oracle with $MEMB_{\widehat{q}}$. Membership oracles play a central role in exact learning in the style of Angluin [3]. Note that, just like the other two algorithms, Algorithm M only needs to solve the fitting construction problem: it may return *any* CQ that fits the input examples, not necessarily one that is equivalent to $\widehat{q}$.

Algorithm M is given as Algorithm 3.2. It works essentially in the same way as Algorithm P except that, between any two product constructions, it minimizes the constructed example. With the latter, we mean to drop facts as long as the oracle $\text{MEMB}_{\widehat{q}}$ tells us that the resulting example is still positive for $\widehat{q}$. This relies on the invariant that, during the run of the **foreach** loop in Line 2, all constructed examples e^* are positive for $\widehat{q}$.

It is not hard to see that Algorithm M returns a CQ q such that $q \subseteq \widehat{q}$, i.e., $q(I) \subseteq \widehat{q}(I)$ for all instances I.

Algorithm 3.3: Algorithm B

```
Input  : collection $(E^+, E^-)$ of labeled examples for
         which a fitting CQ exists
Output : a fitting CQ
1 foreach $s = 1, 2, \ldots$ do
2     if there is a fitting CQ of size $s$ then
3         return a fitting CQ of size $s$
```

However, q need not be equivalent to $\widehat{q}$. If q_Π is the CQ returned on the same input by Algorithm P, then $q_\Pi \subseteq q$, but again the two CQs need not be equivalent.

EXAMPLE 3.3. *Consider again the collection of data examples $E = (E^+, E^-)$ from Example 3.1, that is, $E^+ = \{C_2, C_3\}$ and E^- contains the single instance $\{R(a, b)\}$. The output of Algorithm M depends on the choice of the query $\widehat{q}$ used by the membership oracle. If $\widehat{q}$ is the CQ with canonical database C_6, then the output is $\widehat{q}$. If $\widehat{q}$ is $\widehat{q}$:- $R(x, y), R(y, z)$, the output also is $\widehat{q}$. If $\widehat{q}$ is C_n with n a multiple of 6, the output is the CQ with canonical database C_6.*

Algorithm B. We finally introduce Algorithm B which implements in a straightforward way the bounded fitting approach proposed in [16]. It is based on a size-bounded version of the fitting construction problem that also incorporates fitting existence. This is again defined relative to a query language $\mathcal{Q}$:

Size-Bounded Fitting Given a collection E of labeled examples and a size bound $s \in \mathbb{N}$ (in unary), construct a fitting query from $\mathcal{Q}$ of size at most s if it exists and report non-existence otherwise.

Size-bounded fitting tends to be of significantly lower computational complexity than fitting construction and existence without a size bound, Section 4 has details. Algorithm B calls an algorithm for the size-bounded fitting problem on E with increasing size bounds, see Algorithm 3.3. More details on algorithms for the size-bounded fitting problem are given in Section 4.

EXAMPLE 3.4. *Consider once more the collection of labeled examples $E = (E^+, E^-)$ from Example 3.1. Algorithm B outputs a fitting CQ of smallest size. In this case, there is a unique such CQ, which is q :- R(x,y), R(y,z).*

4. RUNNING TIME AND SIZE BOUNDS

We discuss the computational complexity of fitting construction and fitting existence and, closely related, the size that the smallest CQ that fits a collection of examples may have in the worst case.

Fundamental Considerations. It was shown in [40] that the smallest fitting CQ for a given collection of labeled examples is in general of size exponential in the size of the examples. We illustrate this with an example.

EXAMPLE 4.1. *Let p_i denote the i-th prime number (where $p_1 = 2$). For $n \geq 1$, let $E_n = (E^+, E^-)$ be the collection of labeled examples with $E^+ = \{C_{p_2}, \ldots, C_{p_{n+1}}\}$, and $E^- = \{C_2\}$. By the prime number theorem, the total size of the examples in E_n is polynomial in n. However, every fitting CQ has size greater than 2^n. More precisely, let $k = \Pi_{i=2}^{n+1} p_i$. The canonical CQ of C_k fits E_n and no smaller fitting CQ exists. Indeed, every fitting CQ q must contain a directed cycle in order to fit the negative example, and, in order to fit to the positive examples, the length of each directed cycle in q must be a multiple of p_i for $2 \leq i \leq n+1$, and hence a multiple of k.*

It follows immediately that the fitting problem cannot be solved by an algorithm with sub-exponential running time, simply because it does in general not have enough time to output a fitting. Less trivially, the inherent complexity of the problem also precludes the existence of such an algorithm. This is witnessed by the following.

THEOREM 4.2. *The fitting existence problem for CQs is coNExpTime-complete.*

This result was first shown in [40] and later improved to hold for a fixed finite schema in [12]. The upper bound is actually shown by running Algorithm P, whose output q is guaranteed to fit all positive examples, and then checking in coNP whether q fits the negative examples. One can show that this is the case if and only if a fitting exists.

Theorem 4.2 does not preclude the possibility of a fitting algorithm that runs in time polynomial in the size of the input plus the size of the smallest fitting CQ. Recall that we call such an algorithm *weakly polynomial*. Unfortunately, it follows from the results in [17] that there is no weakly polynomial fitting algorithm for CQs unless P=NP. In fact, it is shown in [17] that there is a class of examples $\mathcal{E}$ such that

(i) fitting existence for CQs is NP-hard on collections of examples from $\mathcal{E}$;

(ii) if there exists any fitting for such a collection E, then there is one of size bounded by $p(||E||)$ for some polynomial p;

(iii) CQs can be evaluated in polynomial time (in combined complexity) on $\mathcal{E}$.

Any weakly polynomial fitting algorithm for CQs would allow us to decide the problem in (i) in polynomial time, thus showing P=NP, as follows. Given a collection E of examples from $\mathcal{E}$ that may or may not have a fitting CQ, we run the algorithm. Note that the algorithm's behavior is unspecified in the case that $\mathcal{E}$ has no fitting: it may return something else than a fitting CQ or never terminate. If the algorithm makes an output, then by (iii) we may check in polynomial time if it is a fitting CQ for E and return 'yes' or 'no' accordingly. If the algorithm exceeds the running time of $q(|E| + p(|E|))$, where q is the polynomial running time bound of the algorithm on collections of data examples that are promised to have a fitting and p the polynomial from point (ii), then we know that $\mathcal{E}$ has no fitting CQ and thus may return 'no'.

Restricted Fitting Problems. It is natural to ask whether the complexity of the fitting problem can be reduced by restricting attention to a subclass of CQs.

Fitting existence is ExpTime-complete for tree CQs [25] and thus still far from tractable. The upper bound extends to classes of CQs whose treewidth is bounded by a constant [7]. Smallest fitting tree CQs may even be of double exponential size [14]. For CQs that take the form of a path, fitting existence is still NP-complete and thus intractable [17]. In this case, however, there is always a polynomially-size fitting (if a fitting exists at all). Nevertheless, what was said above about weakly polynomial fitting algorithms applies already to these restricted classes.

Other (rather strong) types of syntactic restrictions, such as limitations on the use of existential variables, determinacy conditions pertaining to functional relations, and restricted variable depth, were proposed and studied in the literature on ILP in order to gain tractability. An overview can be found in [35].

Instead of changing the class of queries, one may also adopt other restrictions. Requiring that all examples are based on the same database instance, and even that every tuple of domain elements from that instance occur as a positive or negative example (cf. Remark 2.4), does not improve the complexity [40, 12]. An interesting variation motivated by the bounded fitting approach that underlies our Algorithm B is the size-bounded fitting problem, see Section 3. However, also this restriction does not bring tractability. It was shown in [27] that the size-bounded fitting problem for CQs is Σ_2^p-complete (over a schema with an infinite number of relation symbols of arity at most three). Even for tree CQs that take the form of a path, size-bounded fitting is still NP-complete [17].

How Our Three Algorithms Fare. Algorithm P runs in single exponential time, which is worst-case optimal in light of the above considerations. Note, however, that Algorithm P also has *best case* exponential running time which clearly makes it impractical. Of course, Algorithm P is also not weakly polynomial—we had argued above that no algorithm can be. Example 5.1 below gives a concrete family of example collections where a constant-size fitting CQ exists, but Algorithm P computes a fitting that has exponential size.

Algorithm M, which may be viewed as a refinement of Algorithm P, has single exponential running time (independently of the choice of $\widehat{q}$). In contrast to Algorithm P, however, it *is* weakly polynomial provided that the CQ $\widehat{q}$ chosen for the oracle $\text{MEMB}_{\widehat{q}}$ is the smallest fitting CQ [17]. The intuitive reason is that, after each minimization step, the size of the example e^*, which represents the current candidate query, must be bounded from above by the size of $\widehat{q}$; this is because we need at most one fact in e^* for each atom in $\widehat{q}$. Note that the result on the non-existence of weakly polynomial algorithms from above does not apply to Algorithm M because of its use of membership queries.

The worst case running time of Algorithm B is even double exponential (and it is not weakly polynomial). In fact, we have seen in Example 4.1 that smallest fitting CQs may be (single) exponentially large. Moreover, the size of the smallest fitting determines the size bound s that Algorithm B needs to solve the size-bounded fitting problem for and, as noted above, for CQs this problem is Σ^p_2-complete. Algorithm B is thus a $\text{NExpTime}^{\text{NP}}$ algorithm and has the highest worst-case complexity among our three algorithms. Interestingly, it is nevertheless a promising approach to the efficient implementation of the fitting problem. This is based on the expectation that, in practical cases, the size of the smallest fitting query tends to not be excessively large.

An implementation of Algorithm B ("bounded fitting") as well as practical experiments have been presented in [16] for the case of tree CQs. In that case, the size-bounded fitting problem can be translated in a natural way into the satisfiability problem of propositional logic which enables an efficient implementation based on SAT solvers. The SPELL system described in [16] shows competitive performance and often outperforms state-of-the-art systems based on refinement operators (which are discussed in Section 9). For unrestricted CQs, one may attempt to replace the SAT solver with a system for answer set programming or disjunctive logic programming, but we are not aware that this has been done in practice. As we will see in Section 7, Algorithm B has the additional advantage of constructing fittings that generalize well to unseen examples.

5. SUCCINCT FITTINGS

One desirable property of a fitting algorithm is *succinctness*: for many applications, one wants to find a fitting query of small size. We saw in Example 4.1 that the smallest fitting CQ for a given collection of labeled examples may be exponentially large in the worst case. Therefore, the best one can hope for is to produce a fitting query of size not much larger than that of the smallest fitting CQ. For fitting algorithms with access to a membership oracle $MEMB_{\widehat{q}}$, even this is to much: we can only hope to produce a fitting CQ of size not much larger than that of (the smallest query equivalent to) $\widehat{q}$. Intuitively, this is because the membership oracle 'guides' the fitting algorithm to $\widehat{q}$ and not to a smaller fitting CQ.

Let us consider Algorithm P (which does not use a membership oracle). The following example shows that it may produce a fitting CQ that is much larger than a fitting CQ of minimal size. In fact, the size of the CQ produced by Algorithm P cannot be uniformly bounded by any function in the size of the smallest fitting CQ.

Example 5.1. *Consider again Example 4.1. We modify the example slightly: for $n > 0$, let $E_n = (E^+, E^-)$ where $E^+ = \{C_{p_2}, \ldots, C_{p_{n+1}}\}$ (as before), and $E^- = \emptyset$. On input E_n, Algorithm P outputs a CQ of size $\Theta(\Pi_{i=2}^{n+1} p_i)$ (and not equivalent to any smaller CQ), whereas there is a single (and thus constant size) CQ that fits E_n for all n, namely q :- $R(x, y)$.*

As we have just seen, Algorithm P does not even produce a fitting CQ of "near-minimal" size. In the context of PAC learning as discussed in Section 7, fittings of near-minimal size play an important role. It will be beneficial to formalize this notion already here:

Occam Property A fitting algorithm has the *Occam property* if the following holds for some $\alpha \in [0, 1)$ and polynomial p: if the input is a collection of examples labeled according to a "target CQ" q_t, then the output is a fitting CQ of size at most $|E|^\alpha \cdot p(|q_t|)$. If $\alpha = 0$, we say that the fitting algorithm has the *strong Occam property.*[2]

In other words, when a fitting algorithm has the Occam property, it outputs a CQ whose size depends polynomially on the size of the target CQ and sublinearly on the number of the examples (if at all). The above definition of the Occam property is designed to apply both to ordinary fitting algorithms and to fitting algorithms that use a membership oracle. In the latter case, the target CQ q_t in the above definition is required to be the CQ used by the oracle. In the former case, we can always assume without loss of generality that q_t is a fitting CQ of minimal size. Consequently, a fitting algorithm without membership oracle has the Occam property if and only if it outputs a CQ of size at most $|E|^\alpha \cdot p(n)$, where n is the size of the *smallest CQ that fits.* The latter is indeed the standard condition used to define Occam algorithms in the literature on computational learning theory (where Occam algorithms with membership oracles are typically not considered).

[2]The terminology "Occam property" is not entirely standard. In the literature, it is more common to talk about an *Occam algorithm*, meaning a fitting algorithm that has the Occam property *and* is weakly polynomial. We already saw that there is no weakly polynomial fitting algorithm for CQs (without oracle) and thus in particular there is no such Occam algorithm. By decoupling the size of the output from the running time of the algorithm, we can more easily acknowledge that there are super-polynomial-time fitting algorithms with the Occam property (as we will see below).

How Our Three Algorithms Fare. Example 5.1 shows that Algorithm P does not have the Occam property. As we will see, this is an instance of a general phenomenon related to extremal fitting CQs, see Example 6.2.

Algorithm B *does* have the strong Occam property: indeed, it is immediate from the definition of the algorithm that the output CQ is a fitting CQ of minimal size. Note that this does not depend on the fact that Algorithm B increases the size bound by 1 in each iteration. In fact, under the scheme $s = 1, 2, 4, 8$ it still has the strong Occam property (with $p(x) = 2x$).

For Algorithm M, it was shown in [17] that it outputs a CQ whose size is bounded by the size of the query $\widehat{q}$ used by the membership oracle. Therefore, Algorithm M has the strong Occam property.

6. EXTREMAL FITTINGS

We have already seen that there may be several non-equivalent fitting CQs for the same collection of data examples, and in fact it is easy to see that there may be infinitely many. As was observed in [14], the fitting CQs always form a *convex set*. More precisely, whenever two queries q_1, q_2 fit a set of labeled examples, the same holds for every query q with $q_1 \subseteq q \subseteq q_2$. Recall that $\subseteq$ denotes the relation of query containment, i.e., $q_1 \subseteq q_2$ means that $q_1(I) \subseteq q_2(I)$ for all instances I. The maximal elements of the convex set of fitting CQs can be viewed as "most-general" fitting CQs while minimal elements can be viewed as "most-specific" fitting CQs. We refer to these, collectively, as "extremal" fitting CQs. When they exist, they can thus be viewed as demarcating the entire set of fitting CQs, in the spirit of the version-space representation theorem used in machine learning [34, Chapter 2.5]. In applications such as ML feature engineering over relational data [30, 8], extremal fitting CQs are particularly natural candidates to consider as features [14].

Most-Specific Fitting CQs. There are two natural ways to define "most-specific fitting CQs" for a collection of labeled examples E: a CQ q is a

- *strongly most-specific fitting* CQ for E if q fits E and for every CQ q' that fits E, we have $q \subseteq q'$;
- *weakly most-specific fitting* CQ for E if q fits E and for every CQ q' that fits E, $q' \subseteq q$ implies $q \equiv q'$.

There can clearly be at most one strongly most-specific fitting CQ up to equivalence, for any collection of labeled examples. In contrast, the existence of multiple weakly most-specific fitting CQs is not excluded a priori. It turns out, however, that also weakly most-specific fitting CQs are unique and in fact the two notions coincide and are characterized by the product of the positive examples.

THEOREM 6.1. *For all CQs q and collections of labeled examples $E = (E^+, E^-)$, the following are equivalent:*

1. *q is a strongly most-specific fitting for E,*
2. *q is a weakly most-specific fitting for E,*
3. *q fits E and is equivalent to the canonical CQ of $\Pi_{e \in E^+}(e)$ (which must then be well-defined).*

Since Algorithm P computes the canonical CQ of the direct product of the positive examples, it in fact produces a most-specific fitting CQ. This also means that most-specific fitting CQs always exist (if any fitting CQ exists). In contrast, Algorithm M and Algorithm B do *not* in general produce a most-specific fitting:

EXAMPLE 6.2. *Recall that for the collections E_n of labeled examples from Example 5.1, there exists a fitting CQ with a single atom, which is thus output by Algorithms M and B. In contrast, we had seen in Example 5.1 that Algorithm P produces a CQ of exponential size. We claim that, in fact,* every *most-specific fitting CQ q for E_n must be of size at least $k = \Pi_{i=2}^{n+1} p_i$. Indeed, let q be any most-specific fitting CQ q and let q' be the query that expresses the existence of a directed cycle of length k. Since q fits E_n and q' is a (strongly) most-specific fitting CQ, it follows that $q' \subseteq q$. Consequently, q' must contain a directed cycle. However, as we argued in Example 4.1, the length of any such directed cycle must be a multiple of k. Therefore, q must be of size at least k.*

More generally, the example shows that no fitting algorithm with the Occam property will always output a most-specific fitting CQ.

Most-General Fitting CQs. For most-general fittings, the story gets more complicated. There are again two natural ways to define "most-general fitting CQs" for a collection of labeled examples E: a CQ q is a

- *strongly most-general fitting* CQ for E if q fits E and for every CQ q' that fits E, we have $q' \subseteq q$;
- *weakly most-general fitting* CQ for E if q fits E and for every CQ q' that fits E, $q \subseteq q'$ implies $q \equiv q'$.

This time, however, the two notions do not coincide. While every strongly most-general fitting CQ is clearly also weakly most-general, the converse fails:

EXAMPLE 6.3. *Consider a schema consisting of three unary relation symbols P_1, P_2, P_3. Let $E = (E^+, E^-)$ where $E^+ = \emptyset$ and E^- consists of the single-fact instance $\{P_1(a)\}$. Then q :- $P_2(x)$ and q' :- $P_3(x)$ are weakly most-general fitting CQs for E and there is no strongly most-general fitting CQ.*

Another natural variation is the following [14]:

- a finite set of CQs $\{q_1, \dots, q_n\}$ is a *basis of most-general fitting CQs* for E if each q_i fits E and for all CQs q' that fit E, we have $q' \subseteq q_i$ for some $i \leq n$.

It is easy to see that a strongly most-general fitting CQ is simply a fitting CQ that forms a basis of size 1, and that every member of a *minimal* basis of most-general fitting CQs is a weakly most-general fitting.

Unlike for most-specific fitting CQs, the existence of a fitting CQ does not, in general, imply the existence of a most-general fitting CQ. For the strong version, this is shown by Example 6.3. For the weak version, we consider the following example [14].

EXAMPLE 6.4. *Let $E = (E^+, E^-)$ be the collection of labeled examples with $E^+ = \{C_1\}$ and $E^- = \{C_2\}$. Clearly, the Boolean CQ q_0 :- $R(x, x)$ fits E. It follows from results in [14] that no weakly most-general fitting CQ exists for E. Intuitively, this is because a CQ q fits E if and only if its canonical instance, viewed as a graph, is not two-colorable. A graph is two-colorable if and only if it does not contain a cycle of odd length. Thus, each fitting CQ for E must contain a cycle of odd length, and for each such CQ q one can construct a strictly more general fitting CQ q' by increasing the length of this cycle.*

Thus, one cannot reasonably require a fitting algorithm to always output a most-general fitting CQ. Still, one may wish for a fitting algorithm that, on inputs for which a most-general fitting CQ exists, produces such a CQ. The "most-general fitting" here may refer to *weakly* most-general fitting CQs or to *strongly* most-general ones, so there are two variants of this requirement. It is not hard to see that the first variant implies the second.

As it turns out, even if we restrict attention to instances of the fitting problem for which a strongly most-general fitting CQ exists, Algorithm P, M and B are not guaranteed to produce one. The following example shows this for Algorithm P.

EXAMPLE 6.5. *For $n > 1$, let T_n be the database instance with $adom(T_n) = \{a_1, \ldots, a_n\}$ consisting of all facts of the form $R(a_i, a_j)$ with $i < j$. Let $E_n = (E^+, E^-)$ where $E^+ = \{C_1\}$ and $E^- = \{T_n\}$. The following two queries both fit E_n:*

$$\begin{aligned} q &\text{ :- } R(x,x) \\ q' &\text{ :- } R(x_1,x_2), R(x_2,x_3), \ldots, R(x_n,x_{n+1}). \end{aligned}$$

Observe that $q \subsetneq q'$. In fact, q is the fitting CQ for E_n of smallest size while it can be shown that q' is the strongly most-general fitting CQ for E_n.

It is easy to see that Algorithm P outputs q on input E. Therefore, Algorithm P does not, in general, output a strongly most-general fitting CQ when such a CQ exists.

More generally, the example shows that no fitting algorithm with the Occam property will always return a strongly most-general fitting CQ whenever it exists. Since Algorithms B and M have the Occam property, it follows that they do not, in general, output a strongly most-general fitting CQ when such a CQ exists.

Algorithms for Most-General Fitting CQs. Since none of our three fitting algorithms produces most-general fitting CQs even when they exist, it is natural to ask whether there are fitting algorithms with this property. The existence problem for weakly/strongly most-general fitting CQs and the problem of constructing them, when they exist, were studied extensively in [14]. Both of these problems turn out to be decidable and we briefly review the core insights underlying the algorithms.

Weakly most-general fitting CQs can be characterized in terms of *frontiers* [14]. A *frontier* for a CQ q is a finite set of CQs $F(q) = \{q_1, \ldots, q_n\}$ with the property that $q \subsetneq q_i$ for all $i \leq n$, and for all CQs q', if $q \subsetneq q'$ then $q_i \subseteq q'$ for some $i \leq n$. Thus, a frontier for a CQ q is a finite complete set of minimal generalizations of q.

THEOREM 6.6 ([14]). *For all queries q and collections of labeled examples E, the following are equivalent:*

1. *q is a weakly most-general fitting CQ for E,*
2. *q fits E, q has a frontier $F(q) = \{q_1, \ldots, q_n\}$ and no $q_i \in F(q)$ fits E.*

This characterization, together with known results regarding the existence of frontiers for CQs [24, 13], was used in [14] to obtain effective algorithms for the existence, verification, and construction problem for weakly most-general fitting CQs. Regarding the computational complexity, we only mention here that the existence problem is ExpTime-complete [14]. It is worth to point out, however, that the known lower bound only applies when we do not require that the collection of labeled data examples E given as an input has a fitting CQ.

Theorem 6.6 also sheds light on the shape of weakly most-general fitting CQs. In fact, it is known that any CQ that has a frontier must be *c-acyclic* [1, 24, 13] which means that every cycle in the incidence graph of the CQ contains an answer variable. By Theorem 6.6, the same is true for weakly most-general fitting CQs. This is crucially exploited by the algorithms in [14].

Strongly most-general fitting CQs and, more generally, finite bases of most-general fitting CQs turn out to be closely related to *homomorphism dualities*, a fundamental concept that originates from combinatorial graph theory and has found diverse applications in different areas, including the study of constraint satisfaction problems, database theory, and knowledge representation.

With $adom(I)$, we denote the set of values used in database instance I. Recall that a homomorphism $h\colon I \to J$ from instance I to instance J (over the same schema) is a function $h\colon adom(I) \to adom(J)$ such that the h-image of every fact of I is a fact of J. We write $I \to J$ to indicate the existence of a homomorphism from I to J. A *homomorphism duality* is a pair of finite sets of instances $(\mathcal{F}, \mathcal{D})$ such that for all instances I, $F \to I$ for some $F \in \mathcal{F}$ iff $I \not\to D$ for all $D \in \mathcal{D}$. This notion can be further refined by relativizing it: a pair $(\mathcal{F}, \mathcal{D})$ being a homomorphism duality *relative to* an instance J is defined in exactly the same way except that only instances I are considered that satisfy $I \to J$.

THEOREM 6.7 ([14]). *For all Boolean CQs $q_1, \dots, q_n$ and collections of labeled examples E, the following are equivalent:*

1. *$\{q_1, \dots, q_n\}$ is a basis of most-general fitting CQs for E*
2. *each q_i fits E and $(\{I_{q_1}, \dots, I_{q_n}\}, E^-)$ is a homomorphism duality relative to $\Pi_{e \in E^+}(e)$.*

This characterization (and an extension of it for non-Boolean CQs), together with known results on the existence of homomorphism dualities, was used in [14] to obtain effective algorithms for the existence, verification, and construction of bases of most-general fitting CQs. Regarding the computational complexity, we only mention here that the existence problem is NExpTime-complete [14]. The lower bound applies even if the collection of labeled data examples E given as an input is promised to have a fitting CQ. Note that since every member of a minimal basis of most-general fittings CQs is a weakly most-general fitting, it must be c-acyclic.

7. GENERALIZATION

By definition, a fitting algorithm constructs a CQ that fits the labeled examples in its input. Ideally, we would like the constructed CQ to correctly predict the label also of *unseen* examples. This can only be expected if those examples are drawn from the same probability distribution (over the space of all examples) as the input examples, and labeled according to the same target CQ. If it is the case, then we can legitimately say that the fitting generalizes from the input. This is formally captured by the framework of *PAC (Probably Approximately Correct)* learning [39]. To give precise definitions, we first introduce some terminology and notation. An *example distribution* is a probability distribution D over the space of all examples (for some fixed schema and arity). Given CQs q, q_t and an example distribution D,

$$\text{error}_{D,q_t}(q) = \Pr_{(I,\mathbf{a}) \in D}(\mathbf{a} \in q(I) \triangle q_t(I))$$

is the expected error of q relative to q_t and D where $\triangle$ denotes symmetry difference. Hence, $\text{error}_{D,q_t}(q)$ is the probability that q disagrees with q_t on any example drawn at random from the example distribution D.

(Polynomial) PAC Property A fitting algorithm has the (polynomial) PAC property if there is a (polynomial) function $f(\cdot, \cdot, \cdot, \cdot)$ such that for all CQs q_t, $\delta, \varepsilon \in (0, 1)$, $m \in \mathbb{N}$ and probability distributions D over examples of size at most m, if the input consists of at least $f(1/\delta, 1/\varepsilon, |q_t|, m)$ many examples drawn from D that are labeled according to q_t, then with probability at least $1 - \delta$, the algorithm outputs a CQ q with $\text{error}_{q_t,D}(q) < \varepsilon$.[3]

Other common definitions of efficient PAC learning do not even demand that the learning algorithm produces a fitting query, but require that the algorithm is weakly polynomial. However, it is known that there is no such efficient PAC learning algorithm for CQs. A detailed discussion can be found in [17], also for subclasses of CQs such as tree CQs and path-shaped CQs which are not efficiently PAC learnable either. This is closely related to the fact that there is no weakly polynomial fitting algorithm for (these subclasses of) CQs, see Section 4.

A fundamental result in computational learning theory states that every fitting algorithm with the Occam property also has the polynomial PAC property, first shown in [9]. This result is usually stated only for fitting algorithms without a membership oracle, but the same holds in the presence of such an oracle. This relies on the fact that, in Section 5, we defined the Occam property in terms of a target CQ rather than a smallest fitting CQ.

Since Algorithm M has the Occam property, we can conclude that it has the polynomial PAC property. The same applies to Algorithm B. Algorithms P, on the other hand, lacks the polynomial PAC property. In fact:

THEOREM 7.1 ([16, 15]). *Let A be a fitting algorithm that either (i) always produces a most-specific fitting or (ii) produces a strongly most-general fitting whenever it exists. Then A lacks the polynomial PAC property.*

The intuitive reason behind Theorem 7.1 is that extremal fittings tend to overfit to the input examples. Most-specific fittings focus too much on the positive examples in the input and tend to incorrectly predict the label of unseen positive examples. Similarly, most-general fittings focus too much on the negative examples in the input. It is worth contrasting Theorem 7.1 with the result from [5, 29] that, for concept classes with finite VC dimension that are intersection-closed, fitting algorithms that produce most-specific fittings have the polynomial PAC property.

[3]We deviate here from standard textbook definitions of the PAC model by including the bound m on the size of examples as an argument to the sample complexity function f. Most concept classes studied in the computational learning theory literature have an example space that consists of examples of bounded size. The example size can then be treated as a constant and thus m can be omitted as an argument to the function f. In contrast, fitting algorithms for CQs can receive arbitrarily large database instances as inputs. The present definition of the PAC property, following [35], allows the number of examples provided to the fitting algorithm to depend on the example size. That is, if the input contains large examples, the fitting algorithm is accordingly given access to more labeled examples. This is not needed for the positive results mentioned below, but it makes the negative results stronger.

8. COMPLETENESS FOR DESIGN

One application of fitting algorithms is in the context of example-based specification ("query by example"): rather than writing a formal specification, the user provides data examples, and the system infers a query through the use of a fitting algorithm. The premise of this approach, which traces back to [41], is that the user has a good grasp of the desired behavior of the query that they are trying to construct, but not necessarily of the query language. In such a setting, it is desirable that the user is indeed able to obtain their intended query as long as they provide a sufficiently comprehensive (finite) set of examples. Whether this is the case, in general, depends on the fitting algorithm, and this is formalized by the following property:

Completeness for Design A fitting algorithm is *complete for design* if for every CQ q, there exists a collection of labeled examples E such that, on input E, the fitting algorithm produces a CQ equivalent to q.

If a fitting algorithm is not complete for design, there are CQs q for which the algorithm will simply never get it right, no matter how many examples are provided.

It is not difficult to see that every fitting algorithm that produces a most-specific fitting CQ, is complete for design. Indeed, it suffices to pick any collection of examples (labeled according to the query q) that includes the canonical example of q. In particular, this shows that Algorithm P is complete for design.

Similarly, it can be shown that every fitting algorithm with the strong Occam property is complete for design. Indeed, let q be any CQ, and E be any collection of examples, labeled according to q, that includes, for every CQ q' of size at most $p(|q|)$ not equivalent to q a labeled examples that q fits but q' does not. The fitting algorithm, on input E, is guaranteed to produce a fitting CQ of size at most $p(|q|)$, which therefore, by construction, must be equivalent to q. Since Algorithm M and Algorithm B have the Occam property with $\alpha = 0$, it follows that both are complete for design.

Completeness for design is also related to the notion of unique characterizability [13]. More precisely, if a class of queries admits unique characterizations (which means that every query is uniquely characterized by a finite set of data examples), then every fitting algorithm for this class is complete for design. There is no implication in the opposite direction. In fact, CQs are not uniquely characterizable (cf. [13]), even though our fitting algorithms are complete for design.

9. REFINEMENT-BASED APPROACHES

A different approach to the fitting problem has been taken in the field of *inductive logic programming (ILP)* which studies the following abstract problem [35]:

> Given a background theory $\mathcal{B}$ and positive and negative examples, find a theory Σ such that $\mathcal{B} \cup \Sigma$ entails all positive examples and none of the negative examples.

Traditionally, this problem is studied for the language of first-order clauses. Thus, the background theory $\mathcal{B}$ is a finite set of clauses, the examples are clauses, and the sought theory can also be a set of clauses. Note that this a very rich problem setting since first-order clauses include, e.g., all Datalog programs. The CQ fitting problem can be viewed as a special case when all examples in the input (E^+, E^-) share the same database I: the background theory is $\mathcal{B} = I$, there is a positive example $R(\mathbf{a})$ for each $(I, \mathbf{a}) \in E^+$, a negative example $R(\mathbf{a})$ for each $(I, \mathbf{a}) \in E^-$, where R is a fixed relation symbol that does not appear in I, and Σ is restricted to be a single non-recursive Horn clause with head $R(\mathbf{x})$.

Most ILP algorithms conform to a common scheme [35]: start with some initial theory Σ, and, while $\mathcal{B} \cup \Sigma$ is not as required, iteratively adapt Σ as follows:

- if $\mathcal{B} \cup \Sigma$ is *too strong* (entails a negative example), generalize Σ, and
- if $\mathcal{B} \cup \Sigma$ is *too weak* (does not entail a positive example), specialize Σ.

Initially, generalization was done by dropping a clause from Σ while specialization was done by adding a clause. However, it was observed that the induced changes to Σ are too coarse. To address this, Ehud Shapiro introduced in a seminal paper *refinement operators*, which specialize, respectively generalize a given clause [37]. Applied to the task of finding a fitting CQ (or single Horn clause), this amounts to navigating the *containment lattice* of CQs, whose investigation goes back at least to the 1970s [36].

Formally, an *upward (resp., downward) refinement operator* is a function $\rho : \mathrm{CQ} \to 2^{\mathrm{CQ}}$ such that $q \subseteq q'$ (resp., $q' \subseteq q$) for every $q \in \mathrm{CQ}$ and $q' \in \rho(q)$. Thus, an upward refinement operator returns a set of more general queries, while a downward refinement operator returns a set of more specific queries. Intuitively, a refinement operator induces a graph whose vertices are CQs (equivalent CQs form a single vertex) and in which there is an edge from q to q' iff $q' \in \rho(q)$. ILP style fitting algorithms will then search this graph using some strategy. Algorithm 9.1 depicts a template for such an algorithm based on an upward refinement operator ρ and a prioritization strategy S that determines which query to consider next in search. It starts with the most specific query possible and maintains a priority queue of queries to be visited. In each round it selects the query with the highest priority and, if it does not fit, adds its refinements to the queue, prioritized by S. Natural prioritization strategies include breadth-first search (query gets lower priority than all previously seen queries) and accuracy-based

Algorithm 9.1: Algorithm R, parameterized by refinement operator ρ and prioritization strategy S

```
Input  : collection $(E^+, E^-)$ of labeled examples
Output : fitting CQ or "None exists"
1 $q_0$ := canonical CQ of $e^k_\top$ for $k$ the arity of $(E^+, E^-)$;
2 PriorityQueue pq := $\{q_0\}$;
3 while not pq.isEmpty() do
4     $q$ := pq.pop();
5     if $q$ fits $(E^+, E^-)$ then return $q$;
6     Insert every $p \in \rho(q)$ into pq prioritized with $S$
7 return "None exists"
```

strategies (the priority is the accuracy of the considered query over (E^+, E^-)). There is a natural counterpart of Algorithm 9.1 based on downward refinement operators.

It has been observed that, to make Algorithm 9.1 a complete and terminating algorithm for the CQ fitting problem, the refinement operator ρ has to be ideal, which we define next. An upward refinement operator ρ is called *proper* if $q' \not\subseteq q$ for every $q' \in \rho(q)$ and all $q \in$ CQ, *finite* if $\rho(q)$ is finite for every $q \in$ CQ, and *complete* if for every $q_1, q_2 \in$ CQ with $q_1 \subseteq q_2$, there is a sequence $p_1, \ldots, p_n$ of CQs with $p_1 = q_1$, $p_n = q_n$, and $p_{i+1} \in \rho(p_i)$ for all i with $1 \leq i < n$. It is *ideal* if it is proper, finite, and complete. Ideal downward refinement operators are defined similarly.

There is an intimate connection between ideal refinement operators and frontiers as defined in Section 6. We call them *upward frontiers* here and use the dual notion of *downward frontiers*, defined as expected. Frontiers are also called *covers* in the ILP literature. It has been shown that the existence of finite frontiers is a necessary condition for the existence of ideal refinement operators, and that finite frontiers do not always exist, both in the upward and the downward case [19, 35]. This holds in particular for CQs, and even more so for the broader classes of theories Σ that ILP systems aim to support. Thus, in practice one has to compromise, either by dropping one of the three properties (finiteness, completeness, properness), or by restricting the query class. Most ILP systems use an incomplete refinement operator together with heuristics [35, 21]. As our focus is on complete algorithms, we discuss below restrictions of the query class.

A class of queries which enjoys finite frontiers is the class of tree CQs. Indeed, every tree CQ has a polynomially sized upward frontier [6] and an exponentially sized downward frontier [32, 31]. Moreover, for every q, q' with $q \subseteq q'$, there is only a finite number of queries p with $q \subseteq p \subseteq q'$ [31]. Thus, the function returning the frontier of a query *is* an ideal refinement operator, both in the upward and downward case. Other classes of CQs that have finite frontiers were studied in [13, 26]. It is not known whether they admit ideal refinement operators.

How Algorithm R Fares. A general classification in terms of our desired properties is difficult, since the behavior depends on the refinement operator and prioritization strategy used. We discuss only preliminary observations for tree CQs. On the positive side, if the downward frontier is used as a refinement operator, combined with breadth-first search, the downward version of Algorithm 9.1 will always return a weakly most-general fitting (when it exists) [16]. Of course, this also means, by Theorem 7.1, that it does not have the PAC property. This can be fixed by extending the refinement operator in a suitable way to achieve that the resulting algorithm has the Occam property [16]. On the negative side, the length of refinement paths along upward frontiers is not bounded by an elementary function [31], which compromises efficiency for the upward version of Algorithm 9.1.

An ideal downward refinement operator for tree CQs developed in [32] has been implemented in the ELTL incarnation of the DL Learner suite [11]. Since the prioritization strategy used there is quite involved, it is hard to analyze whether ELTL satisfies our desired properties. Experiments from [16] show that ELTL generalizes well to unseen examples, which might be explained by the fact that its prioritization strategy takes the query size into account and gears search towards smaller CQs.

10. SUMMARY AND OUTLOOK

We identified a list of desirable properties of fitting algorithms and their structural interactions (cf. Figure 1), and used them to compare three fitting algorithms for CQs (cf. Table 2). As mentioned in Section 4, Algorithm B was successfully implemented and shown to perform competitively for the special case of tree CQs [16].

Our general motivation comes from the development of *interactive, example-aided methodologies for the synthesis, refinement, and debugging of database queries.* The fitting problem is only one facet of this broader topic. Other facets that require further study include example generation and example-based query refinement.

Extending the scope of our analysis to other query languages remains future work. A detailed analysis of extremal fitting problems for UCQs can be found in [14].

We only considered *exact* fittings. When these are not guaranteed to exist, the fitting problem is perhaps more naturally viewed as a multi-objective optimization problem (with degree-of-fitting as one of the objectives). This is future work but see [8, 28, 18] for some related work.

Acknowledgements. B. ten Cate is supported by the European Union's Horizon 2020 research and innovation programme (MSCA-101031081). C. Lutz is supported by DAAD project 57616814 (SECAI, School of Embedded Composite AI) as part of the program Konrad Zuse Schools of Excellence in Artificial Intelligence.

11. REFERENCES

[1] B. Alexe, B. t. Cate, P. G. Kolaitis, and W.-C. Tan. Characterizing schema mappings via data examples. *ACM Trans. Database Syst.*, 36(4):23:1–23:48, 2011.

[2] B. Alexe, B. ten Cate, P. G. Kolaitis, and W.-C. Tan. Designing and refining schema mappings via data examples. In *Proc. of SIGMOD*, pages 133–144, 2011.

[3] D. Angluin. Queries and concept learning. *Mach. Learn.*, 2(4):319–342, Apr. 1988.

[4] M. Arenas, G. I. Diaz, and E. V. Kostylev. Reverse engineering sparql queries. In *Proc. of WWW*, page 239–249, 2016.

[5] P. Auer and R. Ortner. A new pac bound for intersection-closed concept classes. *Machine Learning*, 66:151–163, 2004.

[6] F. Baader, F. Kriegel, A. Nuradiansyah, and R. Peñaloza. Making repairs in description logics more gentle. In *Proc. KR*, pages 319–328, 2018.

[7] P. Barceló and M. Romero. The complexity of reverse engineering problems for conjunctive queries. In *Proc. of ICDT*, pages 7:1–7:17, 2017.

[8] P. Barceló, A. Baumgartner, V. Dalmau, and B. Kimelfeld. Regularizing conjunctive features for classification. *J. Comput. Syst. Sci.*, 119:97–124, 2021.

[9] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *J. ACM*, 36(4):929–965, 1989.

[10] A. Bonifati, R. Ciucanu, and A. Lemay. Learning path queries on graph databases. In *Proc. of EDBT*, pages 109–120, 2015.

[11] L. Bühmann, J. Lehmann, and P. Westphal. DL-Learner - A framework for inductive learning on the semantic web. *J. Web Semant.*, 39:15–24, 2016.

[12] B. ten Cate and V. Dalmau. The product homomorphism problem and applications. In *Proc. of ICDT*, pages 161–176, 2015.

[13] B. ten Cate and V. Dalmau. Conjunctive queries: Unique characterizations and exact learnability. *ACM Trans. Database Syst.*, 47(4):14:1–14:41, 2022.

[14] B. ten Cate, V. Dalmau, M. Funk, and C. Lutz. Extremal fitting problems for conjunctive queries. In *Proc. of PODS*, 2023.

[15] B. ten Cate, M. Funk, J. C. Jung, and C. Lutz. Extremal fitting CQs do not generalize. *CoRR*, abs/2312.03407, 2023.

[16] B. ten Cate, M. Funk, J. C. Jung, and C. Lutz. SAT-based PAC learning of description logic concepts. In *Proc. IJCAI*, pages 3347–3355, 2023.

[17] B. ten Cate, M. Funk, J. C. Jung, and C. Lutz. On the non-efficient PAC learnability of conjunctive queries. *Inf. Process. Lett.*, 183:106431, 2024.

[18] B. ten Cate, P. G. Kolaitis, K. Qian, and W.-C. Tan. Approximation algorithms for schema-mapping discovery from data examples. *ACM Trans. Database Syst.*, 42(2):12:1–12:41, 2017.

[19] P. R. J. Laag van der Laag and S. Nienhuys-Cheng. Existence and nonexistence of complete refinement operators. In *Proc. of ECML*, pages 307–322, 1994.

[20] S. Cohen and Y. Y. Weiss. The complexity of learning tree patterns from example graphs. *ACM Trans. Database Syst.*, 41(2):14:1–14:44, 2016.

[21] A. Cropper, S. Dumančič, R. Evans, and S. H. Muggleton. Inductive logic programming at 30. *Mach. Learn.*, 111(1):147–172, 2022.

[22] V. Filipetto. Constructing queries from data examples, 2022. MSc Thesis. University of Amsterdam.

[23] G. Fletcher, M. Gyssens, J. Paredaens, and D. Van Gucht. On the expressive power of the relational algebra on finite sets of relation pairs. *IEEE Transactions on Knowledge and Data Engineering*, 21:939–942, 2009.

[24] J. Foniok, J. Nesetril, and C. Tardif. Generalised dualities and maximal finite antichains in the homomorphism order of relational structures. *Eur. J. Comb.*, 29(4):881–899, 2008.

[25] M. Funk, J. Jung, C. Lutz, H. Pulcini, and F. Wolter. Learning description logic concepts: When can positive and negative examples be separated? In *Proc. of IJCAI*, pages 1682–1688, 2019.

[26] M. Funk, J. C. Jung, and C. Lutz. Frontiers and exact learning of $\mathcal{ELI}$ queries under DL-Lite ontologies. In *Proc. IJCAI*, pages 2627–2633, 2022.

[27] G. Gottlob, N. Leone, and F. Scarcello. On the complexity of some inductive logic programming problems. *New Generation Comput.*, 17(1):53–75, 1999.

[28] G. Gottlob and P. Senellart. Schema mapping discovery from data instances. *J. ACM*, 57(2):6:1–6:37, 2010.

[29] S. Hanneke. Refined error bounds for several learning algorithms. *J. Mach. Learn. Res.*, 17:1–55, 2016.

[30] B. Kimelfeld and C. Ré. A relational framework for classifier engineering. *ACM SIGMOD Record*, 47:6–13, 2018.

[31] F. Kriegel. Navigating the $\mathcal{EL}$ subsumption hierarchy. In *Proc. of DL*, 2021.

[32] J. Lehmann and C. Haase. Ideal downward refinement in the $\mathcal{EL}$ description logic. In *Proc. of ILP*, pages 73–87, 2009.

[33] H. Li, C.-Y. Chan, and D. Maier. Query from examples: An iterative, data-driven approach to query construction. *Proc. VLDB Endow.*, 8(13):2158–2169, 2015.

[34] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[35] S. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*, volume 1228 of *Lecture Notes in Computer Science*. Springer, 1997.

[36] G. Plotkin. Lattice theoretic properties of subsumption. Technical report, Edinburgh University, Dept. of Machine Intelligence and Perception, 1970.

[37] E. Y. Shapiro. An algorithm that infers theories from facts. In P. J. Hayes, editor, *Proc. of IJCAI*, pages 446–451, 1981.

[38] Q. T. Tran, C.-Y. Chan, and S. Parthasarathy. Query reverse engineering. *The VLDB Journal*, 23(5):721–746, 2014.

[39] L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27:1134–1142, 1984.

[40] R. Willard. Testing expressibility is hard. In *Proc. of CP*, pages 9–23, 2010.

[41] M. M. Zloof. Query by example. In *Proc. of AFIPS NCC*, pages 431–438. AFIPS Press, May 1975.

ML-Powered Index Tuning: An Overview of Recent Progress and Open Challenges

Tarique Siddiqui Wentao Wu
Microsoft Research
{tasidd, wentwu}@microsoft.com

ABSTRACT

The increasing scale and complexity of workloads in modern cloud services highlight a crucial challenge in automated index tuning: recommending high-quality indexes while ensuring scalability. This is further complicated by the need for these automated solutions to minimize query performance regressions in production deployments. This paper directs attention to some of these challenges in automated index tuning and explores ways in which machine learning (ML) techniques provide new opportunities in their mitigation. In particular, we reflect on our recent efforts in developing ML techniques for workload selection, candidate index filtering, speeding up index configuration search, reducing the amount of query optimizer calls, and lowering the chances of performance regressions. We highlight the key takeaways from these efforts and underline the gaps that need to be closed for their effective functioning within the traditional index tuning framework. Additionally, we present a preliminary cross-platform design aimed at democratizing index tuning across multiple SQL-like systems—an imperative in today's continuously expanding data system landscape. We believe our findings will help provide context and impetus to the research and development efforts in automated index tuning.

1. INTRODUCTION

Automated index tuning improves the performance of databases by recommending indexes that accelerate query execution. There has been extensive research over the past decades [23,30], and *index tuners* have been developed for both commercial and open-source database systems [14, 15, 29, 65].

Figure 1 presents the typical architecture of such an index tuner [14, 15, 65]. It contains three major components: (1) *workload parsing/analysis*, where an input workload (of SQL queries) is parsed and analyzed; (2) *candidate index generation*, which identifies a set of candidate indexes for each query in the workload; and (3) *configuration enumeration*, which searches for an in-

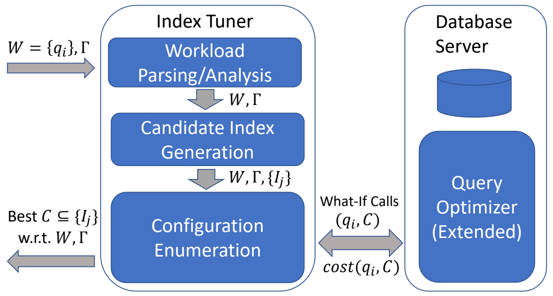

Index Tuner
$W = \{q_i\}, \Gamma$
Workload Parsing/Analysis
W, Γ
Candidate Index Generation
$W, \Gamma, \{I_j\}$
Best $C \subseteq \{I_j\}$ w.r.t. W, Γ
Configuration Enumeration
What-If Calls (q_i, C)
$cost(q_i, C)$
Database Server
Query Optimizer (Extended)

Figure 1: The architecture of an index tuner, where W is the input workload and $q_i \in W$ is a single SQL query, Γ is a set of tuning constraints, $\{I_j\}$ is the set of candidate indexes generated for W, and $C \subseteq \{I_j\}$ represents an index configuration during enumeration.

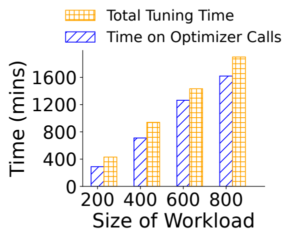
dex configuration from the candidate indexes that meets the user-specified tuning constraints (e.g., the maximum number of indexes allowed or the total amount of storage taken by the indexes) while minimizing the total cost of the workload.[1] For a configuration C considered during enumeration, the index tuner leverages the *what-if API*, an extended functionality of the query optimizer, to estimate the cost of each query on top of C *without* actually building the indexes contained by C [16]. We refer to such query optimizer calls as "what-if (optimizer) calls" in this paper. A what-if call can be time-consuming since it needs to invoke the query optimizer, especially for complex queries.

Despite this success, the recent advances in data management have highlighted the existing challenges and posed new ones. We discuss three key problems.

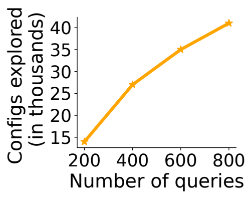
Problem #1: *The growing scale and complexity of database SQL query workloads in modern cloud environments affect the quality of recommended indexes and contribute to increased time, cost, and resource overheads for index tuning.*

Cloud database services, such as Microsoft's Azure SQL Database [1], host millions of databases with large and complex query workloads. Automatically and ef-

[1]A configuration is defined as a set of indexes.



(a) Tuning time



(b) Configurations explored

Figure 2: The growth in tuning time and configuration exploration on increasing workload size.

ficiently tuning indexes at that scale and complexity is a formidable task. In particular, the scalability of index tuning depends on (1) the number of queries in the workload, (2) the number of candidate indexes and resulting configurations that are enumerated, and (3) the number of optimizer invocations or what-if calls. As depicted in Figure 2, we see that the tuning time for a state-of-the-art index advisor [14] grows significantly as we increase the size of the workload. This is primarily because the space of configurations to explore increases (Figure 2b), resulting in a large number of expensive what-if calls (consuming 70% to 80% of the overall tuning time).

Problem #2*: Minimal DBA monitoring and the potential impact on larger workloads in the cloud environments underscores the imperative to mitigate performance regressions stemming due to recommended indexes by index tuners.*

A major impediment to the goal of full automation and scalability is the requirement that index implementations should not cause significant query performance regressions [18]. One important reason for query performance regression (QPR) is that index tuners use query optimizer's cost model (via what-if calls) to measure the improvement in query performance (e.g., execution time) due to recommended indexes [15, 16, 65]. While cost models are much more efficient than directly executing queries, they may not accurately capture the runtime behavior of queries, resulting in a mismatch between the actual and estimated query performance. The issue is further aggravated due to the scale, variety, and complexity of workloads, which make it hard to collect sufficient statistics or incorporate mechanisms for automatically identifying and fixing QPR [18].

Problem #3*: The current approach of building system-specific and tightly-coupled index tuners is less tenable in today's fast-expanding landscape of rapidly growing number and variety of data systems.*

Modern enterprises manage several data systems, each optimized for different use-cases, and frequently add new ones. Data could reside in a variety of locations, e.g., operational stores, data warehouses, or data lakes [3, 46, 47]. Interestingly, only a limited number of database systems, such as Oracle, Microsoft SQL Server, IBM DB2, and PostgreSQL, support index tuning [14, 15, 29, 65]. This is surprising given that the process of index tuning is largely system-independent, with core components such as candidate index generation and configuration search algorithms reusable across systems with minimal changes. Yet, index tuners today are tightly coupled with specific database systems, and developing an index tuner for a new or evolving database system requires massive engineering efforts.

1.1 Paper Overview

In this paper, we reflect on the recent efforts towards addressing the above challenges. While improving the scalability of index tuning and addressing query performance regressions are not new problems, the recent focus has largely been towards leveraging ML-powered techniques that can *efficiently* identify useful configurations without sacrificing the quality of recommendations. Another notable difference compared to prior work is that ML techniques require minimal changes to the underlying query optimizer or to the database system, and can potentially be integrated as "bolt-on" component(s) within existing time-tested and commercially deployed index tuning architectures [14].

Opportunity: *ML-powered techniques have the potential to interoperate with core index tuning components to improve the scalability and reduce query performance regressions, without significant changes to the index tuning architecture, the query optimizer, or the database system.*

Figure 3 outlines an enhanced version of the index tuning architecture depicted in Figure 1 after incorporating ML-based data-driven techniques. It introduces novel software components and functionalities that improve the performance of the end-to-end index tuning workflow: (1) *workload selection* that aims to reduce the size, complexity, and relevance of the input workload; (2) *learned index filter* that aims to prune spurious candidate indexes with little impact on query performance; (3) *MCTS-based enumerator* that aims to improve the effectiveness of index configuration enumeration; (4) *learned cost models* that aim to reduce the number of what-if calls; and (5) *ML-based performance regression predictor* that aims to reduce the chance of query performance regression. We provide an overview of these new functionalities below, and the rest of this paper covers more details of each functionality as well as discussions on the opportunities and open challenges based on lessons learnt from our own experiences.

Workload Selection. We focus on two complementary sub-problems of *workload compression* and *workload*

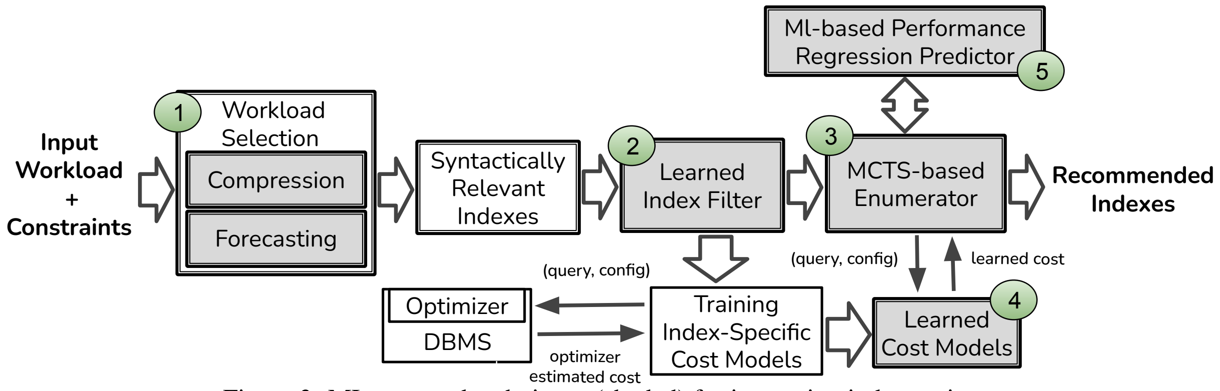


Figure 3: ML-powered techniques (shaded) for improving index tuning.

forecasting. Workload compression selects a small subset of queries from a large input workload, tuning which has the potential to result in as high-quality recommendations as tuning the entire workload. Workload forecasting, on the other hand, predicts arrival rate of queries for *just-in-time* recommendation of indexes, reducing the size of workload that needs to be tuned at given point as well as improving the relevance of recommended indexes for queries in the near future (Section 2).

Learned Index Filter. After selecting SQL queries for tuning, the index tuner parses and analyzes the queries to generate *synthetically relevant indexes* based on *indexable columns* [15] (e.g., columns that appear in filter and join predicates appearing in the *where* clause, as well as columns that appear in the *group-by* and *order-by* clauses). It then tries to identify candidate indexes from the syntactically relevant indexes. Many of such candidate indexes are turned out to be *spurious*, meaning that they have little impact on query performance and can be safely pruned. A learned index filter is developed based on this observation (Section 3.1).

MCTS-based Enumerator. As mentioned, configuration enumeration aims to find the best configuration from the candidate indexes provided. A classic approach to configuration enumeration is *greedy search* [15], which suffers from scalability problems when facing a large search space with many candidate indexes and queries. The MCTS-based enumerator aims to improve the effectiveness of configuration enumeration in large search space by identifying configurations that show promise and potential early on. It leverages reinforcement learning (RL) techniques internally (Section 3.2).

Learned Cost Models. The what-if calls used by index tuner can be expensive, especially when facing large and complex workloads. One important observation we made is that many queries and configurations explored during configuration enumeration are *similar*. This opens up the door of leveraging ML techniques to learn in-situ lightweight cost models for clusters of similar queries and configurations during configuration enumeration, despite the fact that learning a generic cost model is extremely challenging. We can significantly reduce the number of what-if calls by delegating many of them to the cost models learned (Section 3.3).

ML-based Performance Regression Predictor. The what-if calls used by the index tuner rely on the query optimizer's estimated costs, which can be off from the actual query execution time and result in QPR. An ML-based QPR predictor trained on top of query execution data can forecast and therefore avoid QPR firsthand. We highlight the challenges of addressing QPR for production systems, giving an overview of recent efforts and the unsolved challenges that remain open (Section 4).

Cross-platform Index Tuner. Finally, to democratize the ML-powered index tuning techniques over multiple systems, we discuss the problems with the current approach of developing *system-specific* index tuners in today's expanding data system landscape. Towards addressing this, we propose an architecture for a *cross-platform* index tuner, along with abstractions that will allow (the same) index tuning technologies to simultaneously benefit many data systems (Section 5).

1.2 Scope and Limitations

Our primary focus in this paper is on improving the classical *offline* index tuning process as used in commercial tools (e.g., [9, 14, 15, 19, 29, 50, 65, 69]), and the adapted versions of them have also been deployed in modern cloud database services [18]. Notably, there has been significant research efforts on *online* index tuning techniques [6,10,11,41,43,44,48,49,52,53,54], where the index tuner can create/drop indexes *on the fly* to handle workload and data drifts. However, perhaps due to the inherent complexity and variety that comes with dynamic, ad-hoc, and non-stationary workloads, a consensus has not yet been reached on critical open questions of online index tuning such as the architecture, the optimization problem formulation, the optimality guarantee of the recommended indexes, and the performance evaluation criteria. Consequently, to the best of our understanding, such techniques have yet to find substantial adoption in commercial systems.

Meanwhile, there is a line of recent efforts on using ML for holistic database (knob) tuning (e.g., [5, 34, 63, 66, 68, 76, 77, 80]) that goes beyond the scope of index tuning and therefore this paper. There is also lots of related work on using ML for improving other specific aspects or components of database systems, such as physical data layout (e.g., [26, 74]), buffer pool size (e.g., [62]), and query optimizer (e.g., [38, 39, 64, 75, 78]), which we omit in this paper as well.

Moreover, there are common challenges faced by applying ML techniques to solving data management problems that are not restricted to index tuning per se. There has been recent work on addressing such general challenges, such as reducing the overhead of generating training data [67] and dealing with data updates/drifts [32]. An in-depth discussion on these issues is worthwhile but beyond the scope of this paper.

2. WORKLOAD SELECTION

The focus of workload selection has been in two directions: 1) selecting queries to tune, referred to as *workload compression*, and 2) knowing when a query will arrive, referred to as *workload forecasting*. We discuss representative research efforts in each direction.

2.1 Workload Compression

A key factor affecting the scalability of index tuning is the number of SQL queries in the workload. In a typical cloud database service, a workload can contain hundreds or even thousands of queries. Tuning such a large workload in a reasonable amount of time is challenging. It is therefore natural to ask whether index tuning can be sped up significantly by finding a *substitute* workload of smaller size while qualitatively not degrading the result of the application. It is crucial that this compressed workload can be found *efficiently*; otherwise, the very purpose of compression is negated.

Prior workload compression techniques based on sampling and clustering [13, 20] often fail to effectively capture the similarity between queries and miss out less frequent queries that may lead to substantial improvement in performance due to indexes. Furthermore, real workloads have typically more variety in query structures, which makes identifying relevant queries more challenging. To address these issues, we have developed ISUM, an indexing-aware and efficient workload summarization technology [58]. ISUM employs two main techniques to identify relevant queries.

Measuring Potential Improvement: We develop a new technique to efficiently estimate the potential in performance improvement of a query due to indexes *without* requiring optimizer calls, which are key scalability bottlenecks. Our idea is to leverage statistics such as table size, selectivity, and costs of queries while eschewing parts of query optimization unrelated to indexing, to estimate improvement so that it is *highly correlated* with the optimizer estimated improvements.

Capturing Indexing-aware Similarity: On selecting a query, it is also important to quantify the improvement in performance on unselected queries in the workload due to indexes from the selected query. We represent each query as a set of features (derived from *indexable columns* [15]) such that two queries with similar features will likely result in similar sets of indexes. We weigh the features using statistics to capture their relevance to indexes. For instance, features on larger tables are more important, and similarly, the importance of indexable columns can vary depending on whether they occur as part of the filter or join predicates. We can further leverage ML techniques to automatically derive the weights of the features based on table size, selectivity, and position of the columns. Our feature representation also allows us to quantify the similarity between queries with different structures.

Combining both techniques, we measure the improvement due to each query over the entire workload, and develop a *linear-time* algorithm that selects queries in decreasing order of their estimated improvement.

Takeaway #1*: A workload compression technique for scalable index tuning requires efficient estimation of (1) potential performance improvement due to indexes, and (2) indexing-aware similarity between queries, both using minimal optimizer calls (a key scalability bottleneck).*

Figure 4 presents an example of running ISUM on the TPC-DS benchmark workload. Overall, we observe that ISUM can lead to a median of 1.4× and a maximum of 2× performance improvements compared to prior techniques for the same compressed workload sizes. Furthermore, given an input workload consisting of queries along with their costs, the time to select the compressed workload is small (<1%) compared to the tuning time of the compressed workload [58].

Open Challenge #1*: The benefits of shortened tuning time gained from compression is often offset by the overhead involved in parsing queries, gathering statistics, and assessing the improvements brought by recommended indexes across the entire input workload.*

Workload compression techniques, including ISUM, require that statistics such as selectivity, optimizer estimated cost of each query, and other physical plan characteristics are provided as input. We observe that most

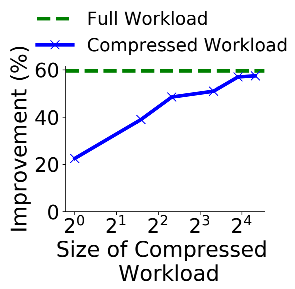


Figure 4: Workload compression on TPC-DS.

database systems expose functionality to collect such information. For database systems where such information is not available, we need to make an optimizer call for each query in the workload, which is expensive for large input workloads.

Open Challenge #2: *Existing workload compression methods focus on specific optimization goals, but there is a need for a more adaptable workload characterization approach that allows for ad-hoc constraints and user-directed query selection.*

Workload compression techniques use pre-defined criteria for selecting queries. However, in practice, one may also want to obtain a representative subset with varying constraints, e.g., 100 most expensive queries while ensuring that every table in the database occurs in at least 5 queries, consuming at least a certain fraction of resources such as CPU and I/O. Thus, the specification for picking a representative subset of a workload depends on the task at hand and requires varying criteria and optimization goals. Additionally, it is crucial to characterize compressed workloads for interpretability. One direction to explore is to report the estimated improvement and drill-downs on how each query in the compressed workload represents queries in the workload that *were not tuned*. Altogether, tighter integration of workload characterization mechanisms into a traditional index tuning engine and their evaluation for a broader set of tasks is an interesting area for future work.

2.2 Workload Forecasting

Workload forecasting allows index tuners to make just-in-time recommendations for the workload expected to arrive in near future. Furthermore, workload forecasting can reduce the number of queries that index tuners need to analyze in each cycle.

As one of the representative works, Ma et al. [37] develop a workload forecasting technique and leverage it to improve index tuning. It uses a two-phase framework. In the first phase, raw queries are pre-processed and clustered based on query templates (i.e., query instances without parameter binding). Clustering is necessary, as it is computationally infeasible to build models to capture and predict the arrival patterns for each template. In the next phase, an ML-based forecasting model is trained for each cluster that predicts how many queries the application will execute in the future (e.g., one hour from now, one day from now, etc.).

Takeaway #2: *Predicting arrival rates of queries in near-future can help reuse traditional offline index tuners for scalable and just-in-time index selection.*

Workload forecasting partially mitigates the inability of offline index tuning in handling dynamic workloads (a core focus of online index tuning [11]) while reusing the offline index tuners. The empirical findings show that when using forecasting, the throughput and latency of MySQL executing real workloads improve by 5× and 78% over the 16-hour period when the indexes are added or removed after every hour. Similarly, over PostgreSQL, the technique achieves 180× better throughput and 99% better latency [37].

Open Challenge #3: *A more holistic forecasting of future workloads, combining both arrival times as well as query instances (e.g., predicate values), is desired to enhance the quality of index recommendations.*

Prior work on index tuning as well as workload forecasting assumes that the query expressions remain unchanged over time. However, the recommended indexes may be sub-optimal when the expressions themselves evolve over time, e.g., a recurring analytical query that looks at last two days of sales data, or a query template that changes bindings based on the day the query runs or the same query template used by different teams with different parameter bindings. Our analysis of enterprise workloads shows that while literal values may change over time, there are high-level patterns that can be learnt to predict the potential bindings in advance. Thus, an interesting direction for future work is to predict entire query instances in addition to the arrival times. There has also been recent work along the line of *robust index selection* [51], with the idea of selecting indexes that are optimal considering the dynamic nature of the workload, which can be combined with workload forecasting to yield even better indexes.

3. SPEEDING UP INDEX TUNING

Searching for the best configuration in a large space with many candidate indexes is inherently challenging. In fact, even a restricted version of the index selection problem is *NP-hard* [17] and/or even *hard to approximate* [12]. State-of-the-art index search algorithms, such as the *greedy* algorithm [14, 15, 30], therefore rely on heuristics to reduce the search space. However, scalability and efficiency remain challenging even in such reduced search spaces. We discuss how we can take a data-driven perspective by leveraging ML techniques to

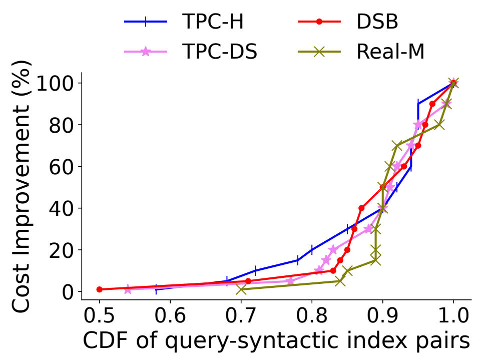


Figure 5: Cost improvement for different fraction of query and syntactically relevant index pairs.

speed up different components of index tuning.

3.1 Filtering Spurious Indexes

Index tuners perform syntactic analysis (e.g., using a set of rules) to select an initial set of indexes for each query, called *syntactically-relevant indexes*, for evaluation [15]. However, as showcased in Figure 5, we observe that 60% to 70% of such indexes are *spurious*—they actually do not result in significant improvement in query performance [59]. Thus, these spurious indexes can be filtered out and the optimizer calls made on these indexes can be avoided.

To prune such indexes early in the search process, we learn a *workload-agnostic* model that uses structure and statistics information in the input (query, index) pair to identify when the index may not lead to a significant improvement in cost [59]. We then use this model to remove a large number of spurious indexes. Our key insight is that we can probe the original physical plan of the query (i.e., the plan generated with existing indexes) to estimate the potential for improvement in the cost of the query due to a given index. For instance, if a join or sort operation is already efficient due to extensive filtering from earlier operations, adding an index that optimizes this operation is less beneficial. Similarly, if a filter column is not selective, we can easily prune an index that uses it as the leading key column. Furthermore, in many cases, we can compare the ordering of physical operators in the original plan with the structure of the index to identify spurious indexes. Altogether, we capture many such signals and train a regression model to automatically learn rules to predict spurious indexes.

Takeaway #3*: Many syntactically-relevant indexes do not lead to improvement in performance. ML models trained on top of domain-specific signals can filter such spurious indexes in orders of magnitude less time compared to making what-if (optimizer) calls.*

As shown in Figure 6, we find that index filtering models can be accurately learnt using (query, index) pairs

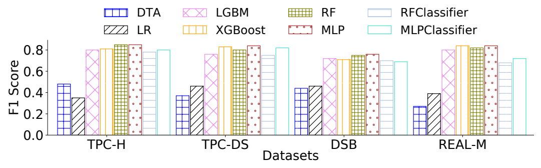


Figure 6: Learned Index Filter.

generated from 3 to 4 databases and workloads and can remove more than 70% spurious indexes with a low rate (typically less than 10%) of false negatives [59].

3.2 Search by Reinforcement Learning

Given the large number of possible index configurations during configuration enumeration for cloud-scale workloads, it is practically impossible to have one what-if optimizer call for *every* configuration and *every* query enumerated. This raises a trade-off between *exploration* (of new configurations) and *exploitation* (of promising configurations that are already known) when determining which configurations are worth what-if calls. We develop a new index search framework based on Monte Carlo tree search (MCTS) [72], a classic reinforcement learning (RL) technology [8, 61], to make better decisions on this exploration/exploitation trade-off. In particular, we adapt the classic greedy search algorithm, typically used during configuration search [14], to handle the trade-off in a data-driven manner as follows:

- **Exploitation:** We can expand configurations that *show promise*, e.g., ones that contain the best configuration found by the greedy algorithm so far as a subset;
- **Exploration:** We can consider configurations that have been overlooked but may have *potential* for improvement, e.g., ones that are not the *winner* configuration found by the greedy algorithm, but have similar costs and can be utilized by more queries.

From this viewpoint, the existing greedy search approach can be viewed as one extreme—it relies on *full* exploitation of what has been found with *no* exploration. Our RL-based approach, on the other hand, encourages more exploration, offering a principled way of tackling the above exploitation/exploration trade-off.

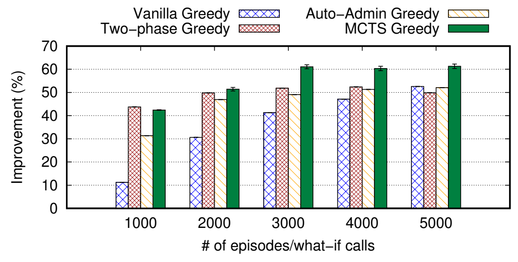
Takeaway #4*: RL-based techniques help navigate exploration and exploitation trade-offs more effectively on deciding which (query, configuration) to evaluate next.*

Figure 7 presents evaluation results on the TPC-DS benchmark and a customer workload (Real-M) with the maximum desired configuration size K set to 20. We compare the MCTS-based approach with both the *vanilla* greedy search algorithm and its variants (shown as *two-phase* greedy and *AutoAdmin* greedy in Figure 7) proposed in [15] and used in the Database Tuning Advisor (*DTA*) developed for Microsoft SQL Server [14], which represents the current state of the art [30]. As depicted in the figure, MCTS outperforms the greedy search algorithms consistently on both workloads w.r.t. the varying number of what-if calls.

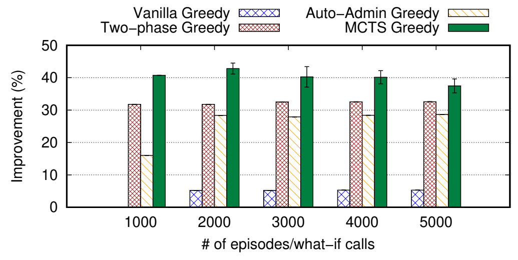
***Open Challenge #4**: Integrating MCTS-based search into commercial index tuning tools such as DTA remains an open problem, considering additional requirements such as anytime tuning, incremental handling of input workloads, and supporting reproducibility (difficult due to randomness inherent to MCTS).*

When the input workload is large and/or complex, we may want to run index tuning with a specific time-bound [14], or we may want to stop the tuning after some time without specifying a budget initially. Therefore, the search algorithm is desired to have the *anytime* property, i.e., it should progressively find better configurations over time. This also requires incremental handling of more queries as input to the search algorithm and maintaining and reasoning about the intermediate state to minimize redundant work. Furthermore, the final recommended indexes can vary due to randomness in MCTS/RL, which affects reproducibility. Handling these challenges in a commercial tuning tool like DTA requires non-trivial adaptations to the MCTS algorithm.

We note that there has been other recent work on ML-based configuration search [31, 33, 44, 45, 55], primarily targeting an online index tuning scenario. This line of work may be adaptable to offline index tuning but it shares the same challenges, as highlighted above, when it comes to integration with existing index tuners. Notably, the recent work by Kossmann et al. [31] proposed training an RL agent that can be used for offline index tuning, where test workloads are presumably similar to training workloads observed by the RL agent. Whether this approach can be further extended to tune completely unseen workload remains an open question.



(a) TPC-DS ($K = 20$)



(b) Real-M ($K = 20$)

Figure 7: Evaluation of MCTS configuration search.

3.3 Reducing What-If Optimizer Calls

To achieve the best possible improvement in performance, the number of optimizer calls made during index configuration search can remain considerable despite pruning of spurious indexes and judicious enumeration of configurations. To further improve the efficiency, we find that a significant number of optimizer calls for costing (query, configuration) pairs can potentially be replaced by more efficient data-driven cost models.

Developing a general cost model that is independent of databases and workloads is hard due to the large varieties in the schema, query structures, and data distributions, despite the intensive efforts in the past decade (e.g., [4,24,35,40,42,57,60,70,71,73,79]). Our key observation to developing a lightweight cost model in the specific context of index tuning is that many queries in large workloads are *self-similar*, e.g., multiple instances of the same stored procedure or query template parameterized differently. Many indexes explored during tuning can also be similar (e.g., sharing the same prefix of key columns, or influencing the same set of operators in the plan), which leads to similar configurations and results in similar cost reductions. As a result, the number of unique cost values is often much smaller than the number of index configurations explored during tuning (e.g., on average only 6 unique costs over 81 configurations explored per query for the TPC-H workload).

To leverage these characteristics, we group similar queries with the same query template and then learn a cost model for each group [59]. For efficient in-situ learning during index configuration enumeration, we develop an *iterative* training procedure (with optimality guarantees) and select diverse training instances (e.g., queries with different selectivities, indexes affecting different operators in the query, etc.) that minimize the number of optimizer calls for training each cost model (e.g., less than 50 optimizer calls per model on average across workloads). We show that it is possible to use low-overhead ML models that are significantly more efficient than making what-if optimizer calls. A key characteristic of these models is that they are *agnostic* of the search algorithm (thus can be used by any algorithm), and do not require changes to the query optimizer.

***Takeaway #5**: ML-based cost models can be used as a generalized cache for similar (query, configuration) pairs, thereby avoiding many "similar" what-if calls.*

Figure 8 depicts the effectiveness of different ML algorithms when used to train per-template cost models, with tree-based models achieving Q-error as low as 1.2. Furthermore, we find that combing ML-based cost models with filtering models for pruning spurious indexes

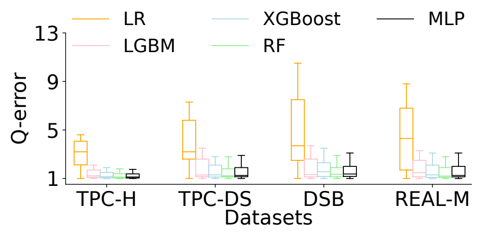


Figure 8: Learned Index Cost Model.

(see Section 3.1) helps scale index tuning to large workloads without sacrificing the quality of the recommended indexes. For instance, for a TPC-DS workload with over 900 queries, combining index filtering and costing models can give index recommendations with similar quality as DTA but in 3× less time.

Open Challenge #5: *Creating ML-based cost models across queries with different templates and across workloads can further improve the scalability by reducing training overhead (i.e., optimizer calls).*

The per-template cost models are less effective for workloads with many templates. There is a significant potential for reducing the number of optimizer calls as well as the number of models if we can generalize cost models across templates. There has been recent work on zero-shot cost models [25]; however, such techniques require a physical query plan (and thus an optimizer call) for featurization. Furthermore, in our current approach, we re-train during every tuning session from scratch due to limited mechanisms for meta-learning or fine-tuning learned models to capture workload and data drifts. This is another area where there are some intersections with online index tuning work [6,44,48].

4. PERFORMANCE REGRESSION

An important requirement of automated index tuning for production systems is creating or dropping indexes should not cause significant query performance regressions (QPR), where a query's execution cost increases dramatically after changing the indexes [21]. Such regressions are a major impediment for fully-automated and scalable index tuning [18]. When an index tuner searches for optimal configurations, it compares estimated improvements of query performance based on the optimizer's estimated costs. Due to well-known limitations in the optimizer's estimates, such as errors in cardinality estimation [27] or cost modeling [71], using the optimizer's estimates can result in significant cost estimation errors. The following trade-off is at the heart of why it is hard to achieve scalability and low rate of QPR in index tuning simultaneously:

Efficiency vs. Accuracy Trade-off: *Optimizer's estimated costs are much faster to compute, but they can be erroneous and result in low-quality recommendations and query performance regressions. On the other hand, actual query execution time is much more accurate but it can only be obtained with significantly higher overhead, affecting the scalability of index tuning.*

One idea to reduce query performance regression is to selectively use execution time during index tuning along with optimizer's estimated cost. Towards this end, Ding et al. [21] proposed a suite of ML techniques that learn over query execution telemetry collected from tens of databases to predict whether or not a new plan due to a selected index configuration has regressed with respect to another plan. Active learning techniques have been used to selectively collect query execution data for ML model training by deploying the same target database on non-production servers [36]. Furthermore, techniques for fixing QPR have also been proposed [21,22].

Takeaway #6: *Leveraging optimizer's estimated costs for index tuning, while* verifying *selected configurations at each step of configuration enumeration through machine learning models trained on query execution statistics, can reduce query performance regressions.*

Unfortunately, from the scalability perspective, the inference process in [21] is expensive since it requires query optimizer calls to obtain the physical plans of the queries. Indeed, the focus of [21] was not scalability, targeting a closed-loop continuous tuning scenario where index tuning time is perhaps trivial considering the workload execution time, especially if there are query performance regressions.

Open Challenge #6: *Detecting configurations that cause query performance regressions with both efficiency and wide coverage, while preserving the scalability of index tuning, remains a significant challenge for large-scale workloads.*

A promising direction, intersecting with challenges discussed in Section 3.3, is to learn pre-trained cost models that bridge the gap between optimizer cost models and the execution behaviour of queries. A challenge that needs to be addressed is that such pre-trained models may not be accurate without requiring plan-level details that need what-if optimizer calls. Toward this end, we can explore techniques similar to the ones used for filtering spurious indexes (Section 3.1) where the original physical plan is probed with properties provided by an index to reason about potential improvement in the cost, as showcased by the very recent work [56]. While learning a generalized model that can work across workloads is challenging (as discussed earlier), we can narrow down the problem by focusing only on indexing-specific improvements. If we can accurately learn such models, it opens up opportunities to eschew both optimizer calls as well as query executions during index tuning, thereby significantly improving the scalability.

5. CROSS-PLATFORM TUNING

The current database management landscape involves many SQL-like systems, with only a few supporting index tuning. While the systems differ in SQL dialects and functionalities (e.g., what-if API), the core ideas for index tuning can often be reused. This is more true for data-driven techniques discussed in this paper, where the ML models have limited dependency on the dialects or unique features of a particular system.

Open Challenge #7: *There are many database systems (where indexes help improve performance) that either have no or low-quality automated index tuning capabilities, forcing users to manually select indexes for their workloads. Adding an index tuner to a new or evolving database system requires substantial engineering overhead, despite that many core ideas in index tuning are cross-platform reusable.*

We hereby call for research efforts on developing a cross-platform index tuner that can work across multiple SQL-like systems, reusing core index tuning techniques (e.g., data-driven ML models as well as the search algorithms currently used in state-of-the-art index tuners). Similar efforts have been made in other areas such as query optimization [7,28]. A cross-platform index tuner needs to adapt to the heterogeneity of features varying across database engines, while reusing the common steps as much as possible. We abstract such a system in Figure 9 with the following main components:

- *Common Data Representation (IR)* consisting of a basic set of elements that need to be captured across systems, e.g., database, tables, columns, logical operators, physical operators, and sub-plans. A cross-language specification such as Subtrait [2] can potentially be leveraged for IR.
- *Common System Interaction APIs* consisting of a common set of APIs that can be used to interact with the database during index tuning. Examples of such APIs include ones for query optimization in the presence of one or more indexes, query execution, creation of hypothetical indexes similar to the what-if API, and building of statistics.
- *Adapters* providing the system-specific implementation of the common system interaction APIs that vary across systems.
- *Index Tuning Planner* enabling cross-platform tuning functionality. It considers system-specific features and user requirements, and outputs an index tuning plan (analogous to query execution plan in database systems). The index tuning task can be represented with a small set of *operators* (e.g., `enumerate`, `combine`, `evaluate`) that can be composed together and configured to perform index tuning. The index tuning plan can be an *acyclic graph* of these operators that is dynamically constructed and optimized by the planner based on system features and user requirements.

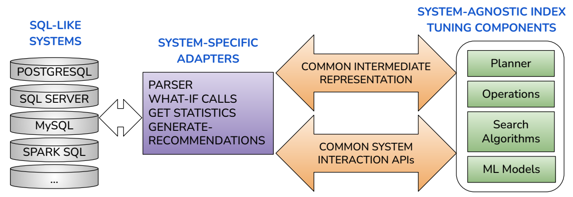


Figure 9: A Cross-Platform Design for Index Tuning.

Overall, a cross-platform index tuner consisting of the components as envisioned above has the potential to democratize index tuning to many more systems than those that are currently supported. In addition, such an index tuner will allow (a) borrowing of the best concepts (implemented as operators) from different index-tuning algorithms, (b) independent improvement and maintenance of the functionality of operators, and (c) extensibility by incorporating new techniques (implemented as new operators) in the future without rewriting the algorithms or changing unrelated operations.

6. CONCLUSION

In this paper, we have highlighted the challenges inherent to automated index tuning, which are further exacerbated within modern cloud environments, and we have discussed recent efforts and opportunities in leveraging ML-powered techniques to address them. We presented an end-to-end analysis of the index tuning workflow, with a focus on the core components such as workload selection and configuration search. We described the issue of query performance regression (QPR) and discussed ML techniques for addressing QPR without affecting index tuning scalability. We also sketched the design of a cross-platform index tuner that extends the current index-tuning software stack to support multiple SQL-like systems. We believe this paper will help create awareness of recent progress and highlight open challenges for future research in index tuning.

Acknowledgement. We thank Surajit Chaudhuri and Vivek Narasayya for their valuable feedback on this work.

7. REFERENCES

[1] Azure sql database. `https://azure.microsoft.com/en-us/products/azure-sql/database/`.
[2] Substrait: Cross-language serialization. `https://substrait.io/`, 2022.
[3] J. Aguilar-Saborit and R. Ramakrishnan. POLARIS: the distributed SQL engine in azure synapse. *Proc. VLDB Endow.*, 13(12):3204–3216, 2020.

[4] M. Akdere, U. Çetintemel, M. Riondato, E. Upfal, and S. B. Zdonik. Learning-based query performance modeling and prediction. In *ICDE*, pages 390–401, 2012.

[5] D. V. Aken, D. Yang, S. Brillard, A. Fiorino, B. Zhang, C. Billian, and A. Pavlo. An inquiry into machine learning-based automatic configuration tuning services on real-world database management systems. *Proc. VLDB Endow.*, 14(7):1241–1253, 2021.

[6] D. Basu, Q. Lin, W. Chen, H. T. Vo, Z. Yuan, P. Senellart, and S. Bressan. Cost-model oblivious database tuning with reinforcement learning. In *DEXA*, pages 253–268, 2015.

[7] E. Begoli, J. Camacho-Rodríguez, J. Hyde, M. J. Mior, and D. Lemire. Apache calcite: A foundational framework for optimized query processing over heterogeneous data sources. In *SIGMOD*, pages 221–230, 2018.

[8] C. Browne, E. J. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. P. Liebana, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *IEEE Trans. Comput. Intell. AI Games*, 4(1):1–43, 2012.

[9] N. Bruno and S. Chaudhuri. Automatic physical database tuning: A relaxation-based approach. In *SIGMOD*, pages 227–238, 2005.

[10] N. Bruno and S. Chaudhuri. To tune or not to tune? A lightweight physical design alerter. In *VLDB*, pages 499–510. ACM, 2006.

[11] N. Bruno and S. Chaudhuri. An online approach to physical design tuning. In *ICDE*, pages 826–835, 2007.

[12] S. Chaudhuri, M. Datar, and V. R. Narasayya. Index selection for databases: A hardness study and a principled heuristic solution. *IEEE Trans. Knowl. Data Eng.*, 16(11):1313–1323, 2004.

[13] S. Chaudhuri, A. K. Gupta, and V. R. Narasayya. Compressing SQL workloads. In *SIGMOD*, pages 488–499, 2002.

[14] S. Chaudhuri and V. Narasayya. Anytime algorithm of database tuning advisor for microsoft sql server, June 2020.

[15] S. Chaudhuri and V. R. Narasayya. An efficient cost-driven index selection tool for microsoft SQL server. In *VLDB*, pages 146–155, 1997.

[16] S. Chaudhuri and V. R. Narasayya. Autoadmin 'what-if' index analysis utility. In *SIGMOD*, pages 367–378, 1998.

[17] D. Comer. The difficulty of optimum index selection. *ACM Trans. Database Syst.*, 3(4):440–445, 1978.

[18] S. Das, M. Grbic, I. Ilic, I. Jovandic, A. Jovanovic, V. R. Narasayya, M. Radulovic, M. Stikic, G. Xu, and S. Chaudhuri. Automatically indexing millions of databases in microsoft azure SQL database. In *SIGMOD*, pages 666–679, 2019.

[19] D. Dash, N. Polyzotis, and A. Ailamaki. Cophy: A scalable, portable, and interactive index advisor for large workloads. *Proc. VLDB Endow.*, 4(6):362–372, 2011.

[20] 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.

[21] B. Ding, S. Das, R. Marcus, W. Wu, S. Chaudhuri, and V. R. Narasayya. AI meets AI: leveraging query executions to improve index recommendations. In *SIGMOD*, pages 1241–1258, 2019.

[22] B. Ding, S. Das, W. Wu, S. Chaudhuri, and V. R. Narasayya. Plan stitch: Harnessing the best of many plans. *Proc. VLDB Endow.*, 11(10):1123–1136, 2018.

[23] S. J. Finkelstein, M. Schkolnick, and P. Tiberio. Physical database design for relational databases. *ACM Trans. Database Syst.*, 13(1), 1988.

[24] A. Ganapathi, H. A. Kuno, U. Dayal, J. L. Wiener, A. Fox, M. I. Jordan, and D. A. Patterson. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In *ICDE*, 2009.

[25] B. Hilprecht and C. Binnig. Zero-shot cost models for out-of-the-box learned cost prediction. *Proc. VLDB Endow.*, 15(11):2361–2374, 2022.

[26] B. Hilprecht, C. Binnig, and U. Röhm. Learning a partitioning advisor for cloud databases. In *SIGMOD*, pages 143–157. ACM, 2020.

[27] Y. E. Ioannidis and S. Christodoulakis. On the propagation of errors in the size of join results. In *SIGMOD*, pages 268–277, 1991.

[28] A. Jindal, K. V. Emani, M. Daum, O. Poppe, B. Haynes, A. Pavlenko, A. Gupta, K. Ramachandra, C. Curino, A. Müller, W. Wu, and H. Patel. Magpie: Python at speed and scale using cloud backends. In *CIDR*, 2021.

[29] A. Kane. The automatic indexer for postgres. `https://github.com/ankane/dexter`, June 2017.

[30] J. Kossmann, S. Halfpap, M. Jankrift, and R. Schlosser. Magic mirror in my hand, which is the best in the land? an experimental evaluation of index selection algorithms. *Proc. VLDB Endow.*, 13(11):2382–2395, 2020.

[31] J. Kossmann, A. Kastius, and R. Schlosser. SWIRL: selection of workload-aware indexes using reinforcement learning. In *EDBT*, pages 2:155–2:168, 2022.

[32] M. Kurmanji and P. Triantafillou. Detect, distill and update: Learned DB systems facing out of distribution data. *Proc. ACM Manag. Data*, 1(1):33:1–33:27, 2023.

[33] H. Lan, Z. Bao, and Y. Peng. An index advisor using deep reinforcement learning. In *CIKM*, pages 2105–2108, 2020.

[34] G. Li, X. Zhou, S. Li, and B. Gao. Qtune: A query-aware database tuning system with deep reinforcement learning. *Proc. VLDB Endow.*, 12(12):2118–2130, 2019.

[35] J. Li, A. C. König, V. R. Narasayya, and S. Chaudhuri. Robust estimation of resource consumption for SQL queries using statistical techniques. *Proc. VLDB Endow.*, 5(11):1555–1566, 2012.

[36] L. Ma, B. Ding, S. Das, and A. Swaminathan. Active learning for ml enhanced database systems. In *SIGMOD*, pages 175–191, 2020.

[37] L. Ma, D. Van Aken, A. Hefny, G. Mezerhane, A. Pavlo, and G. J. Gordon. Query-based workload forecasting for self-driving database management systems. In *Proceedings of the 2018 International Conference on Management of Data*, pages 631–645, 2018.

[38] R. Marcus, P. Negi, H. Mao, N. Tatbul, M. Alizadeh, and T. Kraska. Bao: Making learned query optimization practical. *SIGMOD Rec.*, 51(1):6–13, 2022.

[39] R. C. Marcus, P. Negi, H. Mao, C. Zhang, M. Alizadeh, T. Kraska, O. Papaemmanouil, and N. Tatbul. Neo: A learned query optimizer. *Proc. VLDB Endow.*, 12(11):1705–1718, 2019.

[40] R. C. Marcus and O. Papaemmanouil. Plan-structured deep neural network models for query performance prediction. *Proc. VLDB Endow.*, 12(11):1733–1746, 2019.

[41] B. Mozafari, E. Z. Y. Goh, and D. Y. Yoon. Cliffguard: A principled framework for finding robust database designs. In *SIGMOD*, pages 1167–1182. ACM, 2015.

[42] D. Paul, J. Cao, F. Li, and V. Srikumar. Database workload characterization with query plan encoders. *Proc. VLDB Endow.*, 15(4):923–935, 2021.

[43] R. M. Perera, B. Oetomo, B. I. Rubinstein, and R. Borovica-Gajic. No dba? no regret! multi-armed bandits for index tuning of analytical and htap workloads with provable guarantees. *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[44] R. M. Perera, B. Oetomo, B. I. P. Rubinstein, and R. Borovica-Gajic. DBA bandits: Self-driving index tuning under ad-hoc, analytical workloads with safety guarantees. In *ICDE*, pages 600–611. IEEE, 2021.

[45] R. M. Perera, B. Oetomo, B. I. P. Rubinstein, and R. Borovica-Gajic. HMAB: self-driving hierarchy of bandits for integrated physical database design tuning. *Proc. VLDB Endow.*, 16(2):216–229, 2022.

[46] R. Potharaju, T. Kim, E. Song, W. Wu, L. Novik, A. Dave, P. Pirzadeh, A. Fogarty, G. Dhody, J. Li, V. Acharya, S. Ramanujam, N. Bruno, C. A. Galindo-Legaria, V. R. Narasayya, S. Chaudhuri, A. Nori, T. Talius, and R. Ramakrishnan. Hyperspace: The indexing subsystem of azure synapse. *Proc. VLDB Endow.*, 14(12):3043–3055, 2021.

[47] R. Potharaju, T. Kim, W. Wu, V. Acharya, S. Suh, A. Fogarty, A. Dave, S. Ramanujam, T. Talius, L. Novik, and R. Ramakrishnan. Helios: Hyperscale indexing for the cloud & edge. *Proc. VLDB Endow.*, 13(12):3231–3244, 2020.

[48] Z. Sadri, L. Gruenwald, and E. Leal. Online index selection using deep reinforcement learning for a cluster database. In *ICDE Workshops 2020*, pages 158–161, 2020.

[49] K. Sattler, I. Geist, and E. Schallehn. QUIET: continuous query-driven index tuning. In *VLDB*, pages 1129–1132. Morgan Kaufmann, 2003.

[50] R. Schlosser, J. Kossmann, and M. Boissier. Efficient scalable multi-attribute index selection using recursive strategies. In *ICDE*, pages 1238–1249, 2019.

[51] R. Schlosser, M. Weisgut, L. Hübscher, and O. Nordemann. Robust index selection for stochastic dynamic workloads. *SN Comput. Sci.*, 4(1):59, 2023.

[52] K. Schnaitter, S. Abiteboul, T. Milo, and N. Polyzotis. COLT: continuous on-line tuning. In S. Chaudhuri, V. Hristidis, and N. Polyzotis, editors, *SIGMOD*, pages 793–795. ACM, 2006.

[53] K. Schnaitter, S. Abiteboul, T. Milo, and N. Polyzotis. On-line index selection for shifting workloads. In *ICDE Workshops*, pages 459–468. IEEE Computer Society, 2007.

[54] K. Schnaitter and N. Polyzotis. Semi-automatic index tuning: Keeping dbas in the loop. *Proc. VLDB Endow.*, 5(5):478–489, 2012.

[55] A. Sharma, F. M. Schuhknecht, and J. Dittrich. The case for automatic database administration using deep reinforcement learning. *CoRR*, abs/1801.05643, 2018.

[56] J. Shi, G. Cong, and X. Li. Learned index benefits: Machine learning based index performance estimation. *Proc. VLDB Endow.*, 15(13):3950–3962, 2022.

[57] T. Siddiqui, A. Jindal, S. Qiao, H. Patel, and W. Le. Cost models for big data query processing: Learning, retrofitting, and our findings. In *SIGMOD*, pages 99–113. ACM, 2020.

[58] T. Siddiqui, S. Jo, W. Wu, C. Wang, V. Narasayya, and S. Chaudhuri. Isum: Efficiently compressing large and complex workloads for scalable index tuning. In *SIGMOD*, pages 660–673, 2022.

[59] T. Siddiqui, W. Wu, V. R. Narasayya, and S. Chaudhuri. DISTILL: low-overhead data-driven techniques for filtering and costing indexes for scalable index tuning. *Proc. VLDB Endow.*, 15(10):2019–2031, 2022.

[60] J. Sun and G. Li. An end-to-end learning-based cost estimator. *Proc. VLDB Endow.*, 13(3):307–319, 2019.

[61] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[62] J. Tan, T. Zhang, F. Li, J. Chen, Q. Zheng, P. Zhang, H. Qiao, Y. Shi, W. Cao, and R. Zhang. ibtune: Individualized buffer tuning for large-scale cloud databases. *Proc. VLDB Endow.*, 12(10):1221–1234, 2019.

[63] I. Trummer. DB-BERT: A database tuning tool that "reads the manual". In *SIGMOD*, pages 190–203, 2022.

[64] I. Trummer, J. Wang, D. Maram, S. Moseley, S. Jo, and J. Antonakakis. Skinnerdb: Regret-bounded query evaluation via reinforcement learning. In *SIGMOD*, pages 1153–1170, 2019.

[65] G. Valentin, M. Zuliani, D. C. Zilio, G. M. Lohman, and A. Skelley. DB2 advisor: An optimizer smart enough to recommend its own indexes. In *ICDE*, pages 101–110, 2000.

[66] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM international conference on management of data*, pages 1009–1024, 2017.

[67] F. Ventura, Z. Kaoudi, J. Quiané-Ruiz, and V. Markl. Expand your training limits! generating training data for ml-based data management. In *SIGMOD*, pages 1865–1878. ACM, 2021.

[68] J. Wang, I. Trummer, and D. Basu. UDO: universal database optimization using reinforcement learning. *Proc. VLDB Endow.*, 14(13):3402–3414, 2021.

[69] K. Whang. Index selection in relational databases. In *Foundations of Data Organization*, pages 487–500, 1985.

[70] W. Wu, Y. Chi, H. Hacigümüs, and J. F. Naughton. Towards predicting query execution time for concurrent and dynamic database workloads. *Proc. VLDB Endow.*, 6(10):925–936, 2013.

[71] 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.

[72] W. Wu, C. Wang, T. Siddiqui, J. Wang, V. Narasayya, S. Chaudhuri, and P. A. Bernstein. Budget-aware index tuning with reinforcement learning. In *SIGMOD*, pages 1528–1541, 2022.

[73] W. Wu, X. Wu, H. Hacigümüs, and J. F. Naughton. Uncertainty aware query execution time prediction. *Proc. VLDB Endow.*, 7(14):1857–1868, 2014.

[74] Z. Yang, B. Chandramouli, C. Wang, J. Gehrke, Y. Li, U. F. Minhas, P. Larson, D. Kossmann, and R. Acharya. Qd-tree: Learning data layouts for big data analytics. In *SIGMOD*, pages 193–208. ACM, 2020.

[75] X. Yu, C. Chai, G. Li, and J. Liu. Cost-based or learning-based? A hybrid query optimizer for query plan selection. *Proc. VLDB Endow.*, 15(13):3924–3936, 2022.

[76] B. Zhang, D. V. Aken, J. Wang, T. Dai, S. Jiang, J. Lao, S. Sheng, A. Pavlo, and G. J. Gordon. A demonstration of the ottertune automatic database management system tuning service. *Proc. VLDB Endow.*, 11(12):1910–1913, 2018.

[77] J. Zhang, Y. Liu, K. Zhou, G. Li, Z. Xiao, B. Cheng, J. Xing, Y. Wang, T. Cheng, L. Liu, M. Ran, and Z. Li. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In *SIGMOD*, pages 415–432. ACM, 2019.

[78] W. Zhang, M. Interlandi, P. Mineiro, S. Qiao, N. Ghazanfari, K. Lie, M. T. Friedman, R. Hosn, H. Patel, and A. Jindal. Deploying a steered query optimizer in production at microsoft. In Z. G. Ives, A. Bonifati, and A. E. Abbadi, editors, *SIGMOD*, pages 2299–2311, 2022.

[79] Y. Zhao, G. Cong, J. Shi, and C. Miao. Queryformer: A tree transformer model for query plan representation. *Proc. VLDB Endow.*, 15(8):1658–1670, 2022.

[80] Y. Zhu, J. Liu, M. Guo, Y. Bao, W. Ma, Z. Liu, K. Song, and Y. Yang. Bestconfig: tapping the performance potential of systems via automatic configuration tuning. In *SoCC*, pages 338–350, 2017.

Reminiscences on Influential Papers

This issue's contributors chose influential works that create bridges from other research communities (human-computer interaction, computer architecture, and programming models) to the data management community. Their write-ups highlight the importance of reaching across fields without forgetting the core. 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.

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

Sourav S Bhowmick
Nanyang Technological University, Singapore
`assourav@ntu.edu.sg`

Don Chamberlin in his recent keynote talk at SIGMOD 2023 recalled that one of the goals behind the design of SQL was to make it *"easier to swallow by ordinary people"* and *"easy to understand [..] without any special training"*[1]. Since then 50 years have gone by and SQL has become wildly successful in the corporate world. However, during this time it has also morphed into a query language with many complex query semantics and features. While powerful, research by the Computer Education community has shown that SQL is difficult to understand, learn, and use, diverging from its original goal w.r.t. ease of use. Indeed, one needs specialized training to use SQL and, in fact, this holds true for any declarative query language in the market.

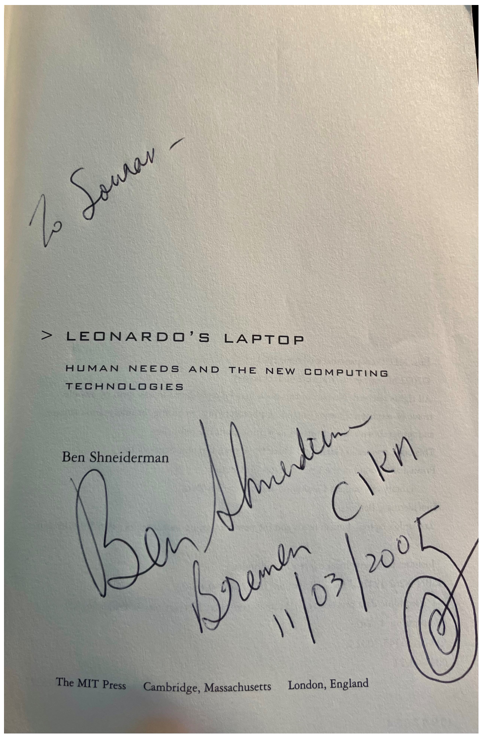
Two decades ago, fresh out from graduate school, I observed that research in data management primarily focused on data structures, algorithms, and performance. Although database usability research started 40 years ago, scant attention was paid to it in practice. Hence, in addition to these traditional issues, I started working on the usability aspects of querying databases as envisioned by Don Chamberlin.

Given that the topic of usability has a strong nexus with human-computer interaction (HCI), I will select two work that influenced my career in a fundamental way - one from the HCI community and another from our Data Management community. Furthermore, I am influenced more by novel, visionary ideas than some specific techniques. So I will diverge from the past authors of this column and select work that are not considered as traditional technical papers - one is a visionary book that influenced me to work on problems that are centered around users and another is a keynote paper that not only influenced my thinking on usability in the context of database systems but also motivated me to persist on my effort on this topic despite initial setbacks and a lack of sufficient attention from the Data Management community.

Ben Shneiderman.
Leonardo's Laptop: Human Needs and the New Computing Technologies.
MIT Press, pages 1-269, 2003.

I bought a signed copy of this visionary, intellectually stimulating, and inspirational book from Ben in CIKM 2005 (Figure 1). It was early days when I was exploring usability and databases. In this book, he asserted that *"the old computing was about what computers could do; the new computing is about what users can do"* and used Leonardo Da Vinci as the inspirational muse to lay down the vision of new computing. Leonardo integrated art with science to serve a practical purpose and produce something that also pleased his patrons. For

[1] https://dl.acm.org/doi/10.1145/3555041.3589336



To Sourav -

> LEONARDO'S LAPTOP

HUMAN NEEDS AND THE NEW COMPUTING TECHNOLOGIES

Ben Shneiderman

[illegible]

Bremen CIKM 11/03/2005

The MIT Press Cambridge, Massachusetts London, England

Figure 1: A signed copy of Leonardo's Laptop.

example, he painted *Mona Lisa*, to please her husband, Francesco del Giocondo, while demonstrating his visual insights and knowledge of geology, plants, and river ecology. So Ben posits that *"technical excellence must be in harmony with user needs"*. We should build products that are usable, useful, and enjoyable.

The book articulated two key steps for realizing the new computing paradigm. First, the promotion of good design w.r.t. the quality of user interfaces and the underlying infrastructure. Second, the promotion of inclusiveness by enabling diverse variety of users, young and old, novice and expert, able and disabled, which Ben referred to as *"universal usability"*. The book also proposed applications of new computing in education, medicine, business, and government.

Back then data management research was primarily about old computing - devising novel data structures and algorithms to demonstrate efficiency and scalability of a software or a technique. Ben's book inspired me to explore about the new computing paradigm in the context of databases. How do we design quality query interfaces and infrastructure to support diverse database user needs? My research on blending visual query formulation and query processing, plug-and-play visual interfaces, user-friendly query visualization abstraction, and understanding and quantifying aesthetics of visual query interfaces are all inspired by the vision of new computing. The vision of universal usability has recently inspired me to explore technologies that can enable learners, young and old, able and disabled, to learn about the topic of relational query processing. This group of users has received scant attention from the Data Management community as research and products primarily target corporate users and developers. It is worth noting that the push for universal usability of data management tools *"makes good business sense because it creates larger audiences for commerce, entertainment, and education"*.

H. V. Jagadish, Adriane Chapman, Aaron Elkiss, Magesh Jayapandian, Yunyao Li, Arnab Nandi, and Cong Yu.

Making Database Systems Usable.

In Proceedings of ACM SIGMOD, pages 13-24, 2007.

This paper is an excerpt from the excellent keynote talk by Jag in SIGMOD 2007, which I attended. As remarked earlier, at that time, the Data Management community primarily focused their attention on data structures, algorithms, and performance issues but not on user-level database usability. This paper brought our attention to the fact that databases are *"hard to design, hard to modify, and hard to query"*. It systematically identifies the pain points encountered by end users and posit that the usability challenges in databases are much more than skin deep. Simply slapping a user-friendly visual query interface on top of a database system does not alleviate these challenges and called for rethinking the underlying architecture of the database system to address them. In particular, it presents the notion of a *presentation data model* as a distinct layer on top of the logical data model. It is envisioned to enable effective personalization and interaction with the database through direct manipulation.

I have used this paper as a mental template for addressing problems related to visual querying. For instance, several of my work focused on blending visual query formulation and processing by exploiting the latency offered by visual query interfaces, which demands rethinking of the underlying query processing component in a visual querying environment. Similarly, our notion of plug-and-play visual query interfaces is inspired by the need of different presentation data model for different users for different data sources to facilitate more effective and efficient visual querying.

Last but not the least, I believe this paper also played a pivotal role in putting the attention of our community back to database usability that was lost for decades. Prior to 2007, I had a hard time publishing papers on usability and data management in the Data Management venues. In fact, I barely squeezed in a short paper in ICDE 2006. In several major venues all the three reviewers would suggest that I should submit these work to HCI venues - as if usability of data systems is not our business! This bleak landscape changed since 2010. Since then, I observed more reviewers in our major venues were open to such work and as a result I was able to publish regularly on this topic. Prior to 2007, often 3 out of 3 reviewers mentioned that usability is not relevant to data management. Nowadays, sometimes it is 1 out of 3 - I believe that is progress! Looking back, I strongly believe that this keynote played a significant role in changing perception and emphasizing *"usability of a database as important as its capability"*.

Carsten Binnig
TU Darmstadt & DFKI, Germany
carsten.binnig@tu-darmstadt.de

Jun Rao and Kenneth A. Ross.
Cache Conscious Indexing for Decision-Support in Main Memory.
In Proceedings of the International Conference on Very Large Data Bases (VLDB), pages 78-89, 1999.

First of all, I would like to thank Pınar for running this column over the last few years. The fact why I like this column is that I am not only learning a lot about past papers of our community but even more I like the fact that the column often tells a very personal story of why a paper was influential to the career of an individual. Moreover, the request of Pınar started my own thought process of which paper I should choose which is an interesting exercise on its own.

To answer this question, I was sitting down and my mind was playing ping-pong with many different ideas. I was happy to discover that there was a way too long list of papers that I had read over my career which inspired me. The bad point was, I had to pick ONE. In the end, I decided to choose the paper "Cache Conscious Indexing for Decision-Support in Main Memory" which appeared in VLDB'99 from Jun Rao and Kenneth A. Ross. I chose the paper due to two reasons.

The first reason is maybe the more obvious one since it was a pick that influenced my career. When I was a PhD, I started to work with Donald Kossmann on database testing which was a highly important topic but at that time not at the core of the database community. After my PhD, I really wanted to shift towards a topic that was more at the core of our community. Around 2008, I started to work on ideas related to leveraging modern hardware for in-memory databases in the context of my (industrial) work on SAP HANA. In this realm, I was reading many papers that contributed to this line of work.

In this body of work, the paper from Ken was starring out. For me personally, it is a "seminal" paper since it early on explored modern CPU architectures with multi-level caches for designing optimal memory-based index structures. The paper looked into the question of how to redesign tree-like in-memory index structures and make them cache-conscious for read-mainly scenarios. In the years after, we saw many papers along similar lines. The paper itself combines a set of simple yet elegant ideas to make an in-memory index cache conscious such as avoiding pointers and using offset computation for the index traversal. As a consequence, data can be kept more densely in the index nodes without "wasting" space for large pointers to child nodes thus making the index cache-friendly.

The result of the paper is an index structure called CSSTree (Cache Sensitive Search Tree) which had later on been followed by the CSB+-tree [1] (Cache Sensitive B^+-Tree) that supports also more write-heavy workloads without giving up the cache-optimal properties. Beyond these topics, Ken has also contributed significantly to a broad range of other important topics regarding databases on modern hardware (e.g., SIMD and GPUs). Today, the whole line of databases on modern hardware is still highly active, and new debates are needed in the context of the cloud given the challenges that hardware is scaling slower than data.

A second reason why I chose the paper is actually because I highly value Ken as a researcher. I only had a few times where I could personally interact with him. One occasion to meet and discuss with him was his visit to Brown University when I was there from 2014 to 2017. These few moments were sufficient to impress me deeply. Firstly, Ken is a very modest and quiet, but extremely knowledgeable person. Secondly, while Ken is known for his solid work on developing database algorithms and data structures for modern hardware, I learned that he also has a second, completely different area of work, which is not even in computer science, but in medicine. Personally, I find this strong ability to research in-depth and breadth extremely fascinating.

[1] Jun Rao and Kenneth A. Ross. "Making B^+-Trees Cache Conscious in Main Memory." In Proceedings of ACM SIGMOD, pages 475-486, 2000.

Peter Alvaro
University of California, Santa Cruz, CA, USA
palvaro@ucsc.edu

Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek.
The Click Modular Router.
In ACM Transactions on Computer Systems, Volume 18, Issue 3, pages 263–297, 2000.[2]

I first read the Click paper in 2009 in Randy Katz's networking seminar at UC Berkeley. At that time, I was a second-year graduate student in Joe Hellerstein's lab, studying data management systems but interested in distributed systems and languages. Riding the tailwind of Boon Loo's dissertation work on "Declarative Networking," I was thinking a lot about transplanting other database ideas (in particular, query languages and dataflow-based execution) into new domains. I was (perhaps narrowly) focused on the idea of "declarative" programming, but I struggled to reconcile the ideology of "what not how" with concerns such as modularity, reuse, and performance. For example, small queries can be quite beautiful, but they become unwieldy as their complexity increases - and programmers tend avoid them because they prevent low-level control of execution.

During my first reading, the Click paper reaffirmed the view of the world I share with a small number of colleagues: that query processing is a rich way to model general computation. As Click shows, routing is a special case of query execution. With the right set of tweaks and extensions, the zoo of routing and switching protocols that we had been studying in class decompose into a set of simple operators (called "elements") that process tuples (i.e., packets) one-at-a-time, alongside a collection of rules about how operators may be composed. An instantiation of a router is hence a dataflow graph that reacts to the insertion of tuples (e.g., those arriving via an external network interface) by outputting tuples (e.g., via an outgoing interface).

Reading more deeply, the paper began to upset my understanding of the role of abstraction in systems programming and its relationship with reuse and performance, teaching me a number of lasting technical lessons. Critical to both the generality and degree of reuse in Click is its ability to combine operators whose interfaces are "pull" (as operators in traditional query processing are, exposing iterators) and those that are "push" (as operators in streaming databases are, fronted by queues). This generality was required to model interfacing with the boundaries of systems where, for example, packets arrive at times outside of our control, and sending packets requires waiting for a device to be ready. Rather than being simply a workaround to handle the edges of the system, however, it makes the programming model surprisingly powerful. The "trick" that makes permits programmers to intermix push and pull operators (but only in ways that are "type-safe," since it does not make sense to directly compose an operator that wants to push to its outputs with an operator that wants to pull from its inputs) is the explicit representation of queues. Rather than hiding them inside streaming-style operators, programmers choose where queues are placed, which influences when scheduling decisions are made. By doing so, Click permits the implementation of operators that (like Eddies [1]) perform scheduling itself.

The Click programming model is not declarative in the sense typically meant by our community. There is no calculus or algebra that gives rise to a space of plans to automatically cost and search: programmers of Click perform query planning by hand. Instead it offers what Kohler[3] called "picture-frame declarativity," allowing implementers to operate freely on both sides of the abstraction. Implementing a new routing protocol is a matter of creating a new query, and can be performed visually, at the level of logical dataflow. Allowing programmers to explicitly place queues gives them control over when scheduling decisions are made, without requiring them to worry about how. Implementing completely new functionality or scheduling policy requires peeling back the abstraction and implementing a new operator in low-level code. Hence Click elements "frame" imperative processing, allowing programmers to enjoy the benefits of high-level programming while maintaining control of the system. Click is fast - faster than I knew a high-level programming model could ever be.

[2]Extends the SOSP 1999 paper from the same authors with the same title.

[3]in a separate talk

It is hard to measure the impact of Click on my own thinking and research. The explicit representation of queues as programmatic constructs rather than hidden plumbing became a key feature of the Dedalus language [2], which became the foundation of my thesis work, some of which departed from pure query languages to target streaming dataflow systems. I have also lost count of how many times I have read the paper. Each time, I find something new. Every database researcher should read it often.

[1] Ron Avnur Joseph M. Hellerstein. “Eddies: Continuously Adaptive Query Processing.” In Proceedings of ACM SIGMOD, pages 261-272, 2000.

[2] Peter Alvaro, William R. Marczak, Neil Conway, Joseph M. Hellerstein, David Maier, and Russell Sears. “Dedalus: Datalog in Time and Space.” In Datalog Workshop, pages 262-281, 2010.

ADVICE TO MID-CAREER RESEARCHERS

We are starting a new series to provide advice to mid-career researchers. There are a number of programs that SIGMOD organizes for researchers at the beginning of their careers (PhD Symposium and the like) and senior people do not (or should not) need much help. There are considerable challenges for those who are about to transition from an early researcher to a more senior role. In academia, these are people who are about to get tenured that comes with starting to think of moving from shorter-term research objectives to longer-term ones. In industrial research, this corresponds to the transition from participating in projects to initiating and leading them. As a community we don't seem to talk about these challenges much. That is the gap this series attempts to fill. We will get the views of senior researchers from diverse backgrounds and diverse geographies. We will continue as long as we find original advice and the views are not repetitions.

M. Tamer Özsu
University of Waterloo

Once upon a Time, in a Computer Engineering Department ....

Tiziana Catarci, Department of Computer, Control and Management Engineering, Sapienza University of Roma
Italy

When I read Tamer's introduction, talking about advice from senior researchers, I had to agree that I am now, indeed in that category. I am presently department chair and used to be vice-rector in the past, acting more like a manager than a researcher. However, I confess that I still have some small personal spaces in which I permit myself to feel like an enthusiastic young post-doc, stop reading documents and start thinking about new research issues. Actually, this is my first advice for those who are in their mid-career: do not just look ahead ("ad maiora" used to say in the ancient Roma), always keep within you the curiosity of the PhD student that you used to be.

When I started my career, everything was slower. We did not have Internet yet (I know, it is difficult to believe), so one had to conduct bibliographical search physically going to the libraries, meet colleagues in person, travel a lot to carry on joint research, at most phone people abroad (not too much because phone calls were expensive) or mail (not e-mail, physical mail) correspondences. Doing research was more complicated than today, the only advantage was that one had a lot of time to think and discuss, much less pressure, and the quality (rather than the quantity) of your publications was the key to academic success. I remember having a lot of fun and strong emotions when the paper envelope you were waiting for from a major conference or journal arrived. With a paper envelope you need to physically get it, open it, take the paper sheet, open it, read it…

Well, I do not want to talk too much about prehistoric times, it is just to establish the scenario where I started, now things are very different also in Italy and laws have been changed several times (everything about academic career here is regulated by national laws, similarly to other European nations but with less autonomy for the universities on the average).

I became assistant professor quite early (before the official completion of my PhD), but then I had to wait a lot for the issuing of a national competition for associate professor, since at that time the only possibility to advance your career was to win a position (in principle not associated with a specific university) among those available through a public call issued by the ministry of universities.

So, my mid-career time came too late to enjoy it, I was almost ready to become full professor, indeed it happened a couple of years later, after a change in the national law, that canceled the global competition and introduced university-level competitions (always regulated as public calls with a comparative evaluation among the applicants and a mostly external committee). It was a pity since mid-career is the time in which you feel your position is more "solid" and you are "in" - so less pressure - but you are supposed to be still in your early academic years, so not much academic commitments, not much "distance" with post-docs and

PhD students, the possibility of creating your own research group and do research having also a lot of fun.

2010: new university law in Italy. The organization of universities in Italy changed a lot with the so-called "Legge Gelmini", that was a sort of revolution, with pros and cons. It impacted the university in several respects, but the two that are more relevant here are the introduction of the national habilitation (similar but not identical to the one in other European countries), the creation of the agency for the evaluation of universities and research (ANVUR), and the change in the recruiting mechanisms.

Recruiting Mechanism The idea behind the recruiting mechanism was to make it closer to the Anglo-Saxon one, with the introduction of two initial positions: a non-tenured track (RTD-A, hybrid between a post-doc and an assistant professor) and a tenured one, RTD-B. RTD-Bs could apply for becoming tenured associate professors after three years with the constraint of having got the national habilitation. Of course, the RTD-B path represents an improvement with respect to the previous situation. Researchers have a clear deadline for getting to the point where they really become faculty members and their mid-career time arrives reasonably early. One problem is that the number of RTD-Bs is not freely established by the universities (or, even better, by the departments), instead it follows complex national rules (too complex and boring to explain here, but I am available for dedicated seminars in case some masochist is curious about them). RTD-As instead are not limited in number because they can be funded with research grants, so rich groups can have many (the reiteration of a traditional inequality of society). This creates the so-called "funnel" problem and many good RTD-As cannot proceed in their academic career.

However, the worst effect on the young people research in scientific sectors derives from the exaggerated importance given to numerical indices both in the university evaluation campaign and in the acquiring of the national habilitation. In order to achieve the habilitation, the RTD-Bs have to (1) overcome the threshold values of the bibliometric medians of their disciplinary fields and (2) conquer a set of "medals" (e.g., being the guest editor of a special issue whatsoever; being the leader of a project whatsoever; being invited to give a PhD course in any university; etc.). Note that the evaluation is carried by strictly referring to a specific scientific sector (SSD) and multidisciplinary research is viewed with suspicion, while it is nowadays considered the key to investigate the "big problems". Therefore, RTD-Bs cannot mainly concentrate on curiosity-driven research, extend their vision, explore synergies with other realms, but they have to (quickly) develop the art of qualification-proofing their CV.

Note that another habilitation must be passed in the mid-career if one wants to apply for a competition for full professorship.

Very recently, in 2022, the recruiting mechanism was changed again canceling the RTD-A figure and introducing a single initial role with tenure, called RTT, which has a maximum of six years to get the habilitation and become associate professor. The RTT positions are limited nationwide, and each university has to comply with a set of constraints that it is not easy to prove satisfiable. Probably the idea was to eliminate the funnel problem, but post-docs have been eliminated as well.

Do not give up! Even if there is now too much bureaucracy in the academia (at least in some countries), and the research pace is getting too fast, nevertheless being a researcher and a professor is one of the best jobs you can get. And this is not mainly because it is an intellectually challenging work, one may meet and collaborate with great minds, and it is possible to work and have fun simultaneously. More importantly, it is possible to give a (small or big) contribution to building a better world.

Professors may not only give disciplinary teachings to students. They might help develop their critical sense and cognitive mechanisms to be able to understand and navigate an increasingly complex world where it is more difficult to distinguish the true from the false.

Carrying on our job, it is possible to advance research with scientific method for a better world (all small contributions count). This is especially important today for the kind of computing research that makes the digital revolution possible.

Ethical Dimension The digital revolution represents an epochal turning point, at least comparable to that which occurred at the time of the industrial revolution in the 19th century, resulting in a disruption of production processes and ways of life. The digital revolution affects the scope of the techniques and tools we use, but it is, first and foremost, a new way of seeing and interpreting the world, implying social, economic, urban, political and many other kinds of changes.

The pervasiveness of the spread of digital tools, particularly those based on Artificial Intelligence, requires ethical reflection by researchers, and by those who develop the systems, those who make them available and those who use them. The ethical dimension has therefore become an essential feature of people doing computer science research, much more so than in the past. Take, for example, data management. We are used to thinking of data as an objective representation of the world, but this is not always the case; data is not the truth, or at least it is not the whole truth. Datasets, no matter how accurate, cannot take a perfect picture of reality and are dependent on humans to acquire, process and store them. Data are socially and politically oriented constructions; making a dataset means defining choice criteria that determine inclusions and exclusions. We need to make sure that the foundational choices are ethical, that the data reflect a worldview that is free of discrimination and attentive to people's well-being. Today this does not always happen: the large amount of data required for training by machine learning systems results in the use of unsupervised data, with all the ethical problems associated with this practice. In this way, artificial intelligence systems end up containing and conveying bias, producing discourses, or making decisions influenced by biases and stereotypes.

Gender Problem Many studies point out the presence of a significant and long-lasting gender gap in the field of computing disciplines. The gender gap exists at all levels: university students, researchers, professors, public administration, and industry. Among STEM disciplines, Computer Science is the one with the highest gender unbalance, with the percentage of women enrolled in ICT graduate programs usually ranging between 10 and 20 percent depending on the country. As a result, the new digital society is designed almost exclusively by men, losing the value of diversity and, consciously or unconsciously, risking reinforcing prejudice. Therefore, it is especially important to encourage girls to undertake studies in digital disciplines and to support those who are at the beginning of their careers or at critical transitions, such as tenure. All of us need to be mentors for those younger ones.

Summary. When you decide to pursue a career in the academy, you may encounter many difficulties, obstacles, and moments of discouragement, but also have great satisfaction, exaltation, and a lot of fun.

By the time you get to the mid-career period, many doubts and problems are behind you (if you are still in the academy and did not give up) and a less stressful time begins with even more freedom to study, create your own group, and choose your own ventures. This is also the time when you may decide to impact the society, not just work for yourself, but do it to contribute to the creation of a better world and be a positive example for the younger generation.

Future Database Engine Development: You Will Only Need One Programming Language

Tianzheng Wang
Simon Fraser University
tzwang@sfu.ca

Database systems must make good use of the hardware for high performance. This is usually done by implementing their core components (storage engine, optimizer and query execution engine) in a low-level programming language (PL) such as C/C++ that can directly "talk to the hardware." But these PLs traditionally lacked high-level abstractions, lowering DBMS developer productivity. Some systems [18, 16, 10] then mix different PLs to balance productivity and performance. For example, Presto [16] and Spark [18] originally used Java but are now replacing their query engines with new ones [12, 1] built in C++ for higher performance. However, doing so brings such non-trivial challenges as interacting with different PL runtimes [1].

Recent advances in PLs, in particular C++ and Rust, have the potential of removing such need. They introduce many desirable features that improve productivity without sacrificing performance. This means core DBMS components could all be implemented in one PL (although other non-performance critical parts may still stick with higher-level PLs). To see why, let us first review the requirements DBMSs pose for PLs.

What does a DBMS Need from a PL?[1] Different DBMS components pose varying requirements on productivity and performance. The optimizer is logic-centric, so there is a strong need for coding productivity to easily express various optimizations. The storage and query engines, however, need to handle parallelism and concurrency by taking full advantage of the hardware, making performance the top priority. Developers are willing to tolerate extra complexity for better performance, such as implementing lock-free indexes [11]. A query engine also needs to implement complex relational algebra and extensions, posing higher requirements on coding productivity than storage engines. As a result, the de facto standard for programming storage and query engines has been traditional C/C++, whereas developers may choose a higher-level PL for optimizers. This has led to the state of mixing PLs for DBMS components.

[1]This paper targets PLs used to implement DBMSs themselves, instead of "database programming languages" which explored the convergence of database systems and PLs [2].

The Case for Modern C++. At a first glance, C++ may be too "low-level" as a native language. However, *modern* C++ (17, 20 and later) comes with features that improve performance and productivity. For example, C++20 allows dynamic memory allocations at compile-time, improving performance by shifting calculations to compile-time and generating smaller binaries [6]. They can also improve productivity as the compiler can detect certain undefined behaviors and leaks at compile-time [7]. In a DBMS, memory used by optimizers typically has a lifespan of queries, yet for storage engines the lifespan is a transaction, requiring different allocators. Some may be a bump allocator that reclaims all memory chunks as a whole, while others should support "real" deallocations of individual chunks. The polymorphic memory resource (`std::pmr`) [17] provides a promising solution, with a set of utilities to manage runtime polymorphism of memory allocations with unified interfaces.

Another example is C++20 coroutines [5] which are being adopted by recent work [9, 13]. Traditionally, query engines implement the iterator model with little PL support and thus an operator must manually maintain states and intermediate data, such as current scan cursor position. C++20 coroutines can help alleviate this problem with a generator-based model. A scan operator's `get_next` can be turned into a coroutine that directly yields each valid row without having to maintain the current row cursor, simplifying implementation.

Future Directions. I believe it is now feasible to satisfy the needs of core DBMS components using a single PL, and many more modern PL features remain to be explored. Beyond C++, Rust [15] and Carbon [4] have gained much attention. It is promising to quantitatively compare these PLs for DBMS implementation. Earlier efforts such as EXODUS [3] and the E programming language [14, 8] have contributed to PL designs. Revisiting them and devising new PLs for DBMS implementation is also promising. Despite the new PL features, compiler and ecosystem support often fall short for serious DBMS development. It is time again for DBMS developers to participate and influence PL and compiler design to push the desired features early into future PLs.

REFERENCES

[1] A. Behm, S. Palkar, U. Agarwal, T. Armstrong, D. Cashman, A. Dave, T. Greenstein, S. Hovsepian, R. Johnson, A. Sai Krishnan, P. Leventis, A. Luszczak, P. Menon, M. Mokhtar, G. Pang, S. Paranjpye, G. Rahn, B. Samwel, T. van Bussel, H. van Hovell, M. Xue, R. Xin, and M. Zaharia. Photon: A fast query engine for lakehouse systems. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 2326–2339, New York, NY, USA, 2022. Association for Computing Machinery.

[2] M. J. Carey and D. J. DeWitt. Of objects and databases: A decade of turmoil. In *Proceedings of the 22th International Conference on Very Large Data Bases*, VLDB '96, page 3–14, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[3] M. J. Carey, D. J. DeWitt, D. Frank, M. Muralikrishna, G. Graefe, J. E. Richardson, and E. J. Shekita. The architecture of the EXODUS extensible DBMS. In *Proceedings on the 1986 International Workshop on Object-Oriented Database Systems*, page 52–65, 1986.

[4] C. Carruth. Carbon language: An experimental successor to C++. *CppNorth*, 2022.

[5] Coroutines. `https://en.cppreference.com/w/cpp/language/coroutines`, 2022.

[6] A. Fertig. C++20 dynamic allocations at compile-time, 2021. `https://andreasfertig.blog/2021/08/cpp20-dynamic-allocations-at-compile-time/`.

[7] B. Filipek. constexpr dynamic memory allocation, C++20, 2021. `https://www.cppstories.com/2021/constexpr-new-cpp20/`.

[8] E. N. Hanson, T. M. Harvey, and M. A. Roth. Experiences in DBMS implementation using an object-oriented persistent programming language and a database toolkit. In *Conference Proceedings on Object-Oriented Programming Systems, Languages, and Applications*, OOPSLA '91, page 314–328, New York, NY, USA, 1991. Association for Computing Machinery.

[9] Y. He, J. Lu, and T. Wang. Corobase: Coroutine-oriented main-memory database engine. *PVLDB*, 14(3):431–444, Nov 2020.

[10] M. Kornacker, A. Behm, V. Bittorf, T. Bobrovytsky, C. Ching, A. Choi, J. Erickson, M. Grund, D. Hecht, M. Jacobs, I. Joshi, L. Kuff, D. Kumar, A. Leblang, N. Li, I. Pandis, H. Robinson, D. Rorke, S. Rus, J. Russell, D. Tsirogiannis, S. Wanderman-Milne, and M. Yoder. Impala: A modern, open-source SQL engine for Hadoop. 2015.

[11] J. J. Levandoski, D. B. Lomet, and S. Sengupta. The Bw-Tree: A B-tree for new hardware platforms. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 302–313, 2013.

[12] P. Pedreira, O. Erling, M. Basmanova, K. Wilfong, L. Sakka, K. Pai, W. He, and B. Chattopadhyay. Velox: Meta's unified execution engine. *PVLDB*, 15(12):3372–3384, Aug 2022.

[13] G. Psaropoulos, T. Legler, N. May, and A. Ailamaki. Interleaving with coroutines: A practical approach for robust index joins. *PVLDB*, 11(2):230–242, Oct 2017.

[14] J. E. Richardson, M. J. Carey, and D. T. Schuh. The design of the E programming language. *ACM Trans. Program. Lang. Syst.*, 15(3):494–534, Jul 1993.

[15] Rust Foundation. Rust programming language, 2022.

[16] R. Sethi, M. Traverso, D. Sundstrom, D. Phillips, W. Xie, Y. Sun, N. Yegitbasi, H. Jin, E. Hwang, N. Shingte, and C. Berner. Presto: SQL on everything. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1802–1813, 2019.

[17] `std::pmr::polymorphic_allocator`. `https://en.cppreference.com/w/cpp/memory/polymorphic_allocator`, 2022.

[18] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, USA, 2012. USENIX Association.

Peer-Reviewing Processes and Incentives: Data Management Community Survey Results

Alexandra Meliou
Univ. of Massachusetts Amherst

Sourav Bhowmick
Nanyang Technological Univ.

Karl Aberer
EPFL

Divy Agrawal
UC Santa Barbara

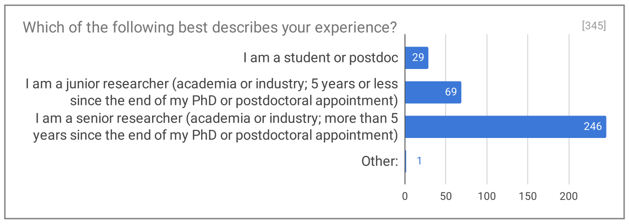
Angela Bonifati
Lyon 1 University, CNRS & IUF

Vanessa Braganholo
Univ. Federal Fluminense

Floris Geerts
University of Antwerp

Wolfgang Lehner
TU Dresden

Divesh Srivastava
AT&T Chief Data Office



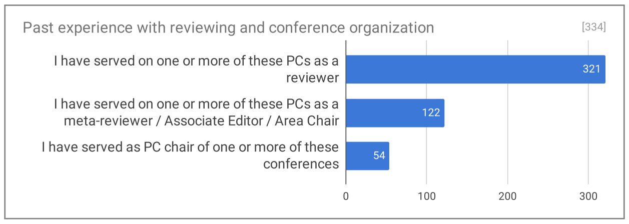


Figure 1: Summary of participants' seniority (left) and reviewing experience (right). Most respondents were senior researchers (71%), and about 16% overall have served in the role of PC chair.

ABSTRACT

Reviewing papers for conferences is an important and hard task that brings several challenges. The Data Management community has been increasingly struggling with high reviewer load, low-quality reviews and low reviewer engagement, unethical reviewing practices as well as undeclared and under-declared conflicts of interest. In this article, we report the results of a survey we conducted to gather the opinion of the Data Management community regarding what could be done to address these challenges. We reached out to about 1,200 members of the data management community with relevant reviewing experience and collected 345 anonymous responses. We plan to follow up with a subsequent report, discussing in more depth particular proposals, inspired by the collective feedback from the community.

1. CHALLENGES AND MISSION

The Data Management research community has worked towards important innovations in our submission and reviewing processes across many of our venues. Examples include the implementation of multiple submission cycles, opportunities for author feedback and revisions, promotion of reproducibility and data sharing, manual checks for review quality, automated COI checks, etc.

However, we also struggle with pain points that have been exacerbated in recent years, as we observe increased reviewer fatigue and declining engagement, as well as challenges with improper conflict declaration. These issues compromise the effectiveness, efficiency, and integrity of our processes. We briefly discuss them here.

- **High reviewer load:** With several deadlines through the year, author feedback phases [2], revision cycles, and participation on multiple PCs, reviewers are often overloaded. The issue is not simply with the number of papers one is called to review, but with the fact that reviewing responsibilities often span the entire year, making it hard for reviewers to plan these around their other career and personal responsibilities.
- **Low-quality reviews and low reviewer engagement:** Our community has been observing an uptick on reviews that are terse, dismissive, and unconstructive. Some reviewers do not respond promptly, or at all, to requests to contribute to discussion, or update their reviews. Late reviews are widespread, reducing the effectiveness of the author feedback and discussion processes. As an example, in the first three submission cycles of SIGMOD 2024, only about 20% of submissions had all three reviews by the review deadline; a little under 20% of submissions were still missing at least one review 5 days after the deadline; about a dozen submissions were still missing reviews 10 days after the deadline.
- **Unethical reviewing practices:** We want to ensure processes that guard against coordination and collusion between authors and reviewers to get papers accepted, resulting in dishonest reviews [3]. Such re-

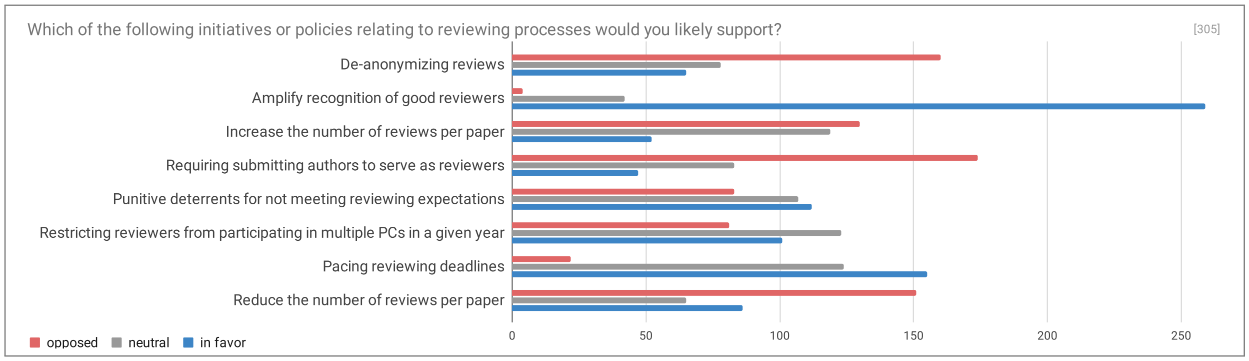


Figure 2: Aggregate responses on the reviewing processes initiatives. There is strong support for amplifying the recognition of good reviewers and for pacing reviewing deadlines; we observe clear opposition to de-anonymization of reviews and requiring submitting authors to serve as reviewers. Other proposals received more split feedback.

views often are of low quality and superficially positive regardless of the content of the papers.

- **Undeclared and under-declared conflicts:** Authors often fail to accurately declare conflicts of interest (COI) with the PC, resulting in burdensome inefficiencies in paper assignment, and potential conflicts in assignments if those are not caught in time. Despite efforts to support conflict entry, grace periods for COI entry, and personalized reminders to authors who fail to complete this task, the problem stubbornly persists.

Several of our executive bodies have called together a task force to collect community feedback and propose policies and initiatives to help address these issues. The joint task force is chaired by Sourav Bhowmick (Nanyang Technological University) and Alexandra Meliou (University of Massachusetts Amherst), and includes the following members: Karl Aberer (Chair of the ICDE Steering Committee), Divy Agrawal (Chair of the ACM SIGMOD Executive Committee), Angela Bonifati (President of the EDBT Executive Board and Association), Vanessa Braganholo (PVLDB Advisory Board), Floris Geerts (Chair of the PODS Executive Committee, ICDT Council member), Wolfgang Lehner (Managing Editor of PVLDB), Divesh Srivastava (President of the VLDB Endowment Board of Trustees).

One of the first initiatives of the task force was the release of a community survey. In this report, we discuss survey participation, present the questions posed, and summarize some aggregate results. The task force plans to work on and release recommendations for possible policies and initiatives in a subsequent report.

2. SURVEY DESCRIPTION AND RESULTS

The survey was advertised by direct email to a list of about 1,200 data management researchers who have served on relevant Program Committees in recent years. We avoided broader advertisement on social media and mailing lists such as DBWorld, as we wanted to keep

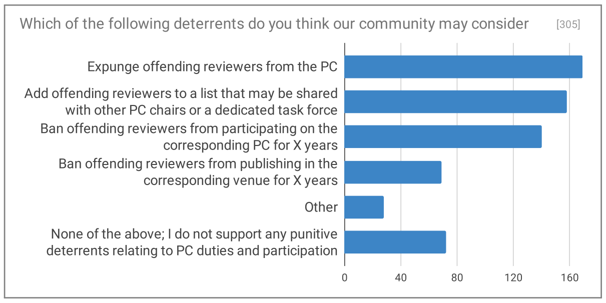


Figure 3: Participant support appears to correlate with the measure's severity. The majority support some punitive deterrent for neglecting PC responsibilities, but 24% of respondents oppose any penalties.

the survey audience targeted to researchers who have had some experience with reviewing tasks in data management venues. The landing page of the survey introduced participants to the objectives of the survey, summarizing the bullet point list in Section 1. The survey made responses optional, meaning that participants could choose not to answer some of the questions. The survey did not collect identifying information from the participants, and we obtained Institutional Review Board (IRB) approval for processing and analyzing the results.

A total of 379 people engaged with the survey in some capacity (i.e., they at least clicked on the link) and 345 submitted answers to at least one question. The survey organized questions in the following sections: (1) general information on the participants' experience; (2) feedback on reviewing process policies; (3) feedback on submission policies; (4) expectations on the role of Associate Editors and meta-reviewers; (5) feedback on reviewing culture initiatives; and (6) general free-text feedback. In the discussion of each question, we will report the number of participants who engaged with the question and submitted responses.

2.1 General Information on Participants

The first section of the survey requested information on

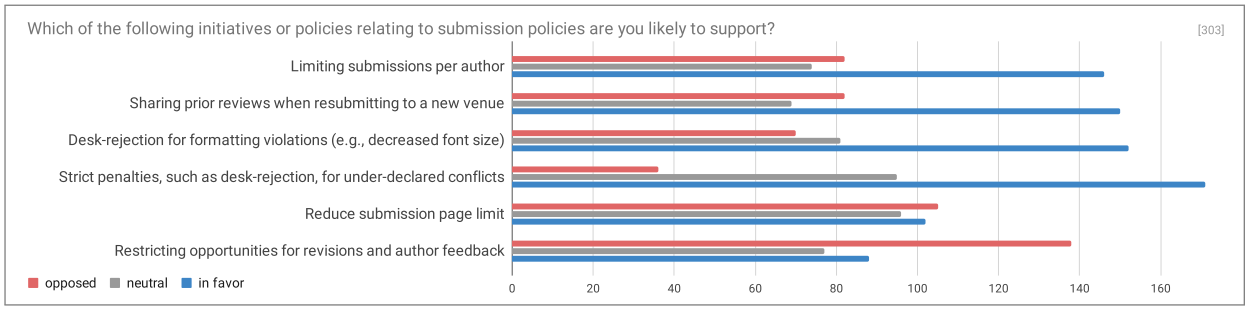


Figure 4: Respondents were against reducing opportunities for revisions and author feedback, and they were split on the idea of reducing submission page limits. All other proposals received majority support, with the strongest support indicated for strict penalties for under-declared conflicts.

participants' seniority and experience in reviewing and conference organization. The question on experience specified a list of data management venues (SIGMOD, VLDB, PODS, ICDE, EDBT, ICDT), and asked participants to indicate whether they have served in reviewing, meta-reviewing, or PC-Chair roles (multiple answers could be selected). A total of 345 and 334 participants responded to the seniority and experience questions, respectively. The results are shown in Figure 1. The vast majority of participants identified themselves as senior researchers (71%); about 16% of all respondents have served in the role of PC chair, and about 36% have served in the role of Associate Editor or meta-reviewer.

2.2 Review Processes

In the subsequent section, the survey asked participants to indicate whether they support or oppose possible policies relating to peer-reviewing practices. We included 8 policy suggestions organized in a Matrix table with a Likert scale (opposed, neutral, in favor). To get additional context on each proposed initiative, including likely benefits and possible downsides, we instructed the participants to hover over the information icon ⓘ. We supply the summary information for each policy proposal in Figure 9.

A total of 305 participants responded to this question. We show the aggregated responses in Figure 2. We observe that the community is particularly supportive of amplifying the recognition of good reviewers, but it is more split with respect to punishment for poor reviewing performance (though the majority does support such measures). There is also clear support for pacing reviewing deadlines. Some measures that met strong opposition are de-anonymization of reviews, adding submitting authors to the reviewing pool, and reducing the number of reviews per paper.

Our survey dove deeper into the topic of punitive deterrents with the question "Which of the following deterrents do you think our community may consider implementing to avert problematic behaviors (e.g., late and/or low-quality reviews, unresponsive reviewers)?" This question also received 305 responses, and participant support seems to correlate with the measure's severity: 55% of participants support expunging reviewers from the PC, 51% believe that PC chairs or a dedicated task force should maintain and share lists of problematic reviewers, 46% support banning such reviewers from participating in PCs for a number of years, and only 23% supports banning such reviewers from publishing in the corresponding venue. About 24% of respondents oppose punitive deterrents relating to PC duties and participation. All these results are summarized in Figure 3.

This question also included an option to propose "other" deterrents. A total of 28 respondents entered suggestions for this option. Several emphasized making issues public, but many used this field to highlight nuances in how we judge reviewing performance and when response should be escalated, caution on the sensitivity of maintaining damning information, and suggestions to prioritize training, feedback, and open communication.

2.3 Submission Processes

Our questions on policies and initiatives relating to submission processes were also organized in a Matrix table with the same Likert scale (opposed, neutral, in favor). Again, participants could access additional information for each initiative by hovering over the information icon ⓘ (more details in Figure 9).

A total of 303 participants responded to this question (Figure 4). Respondents were against reducing opportunities for revisions and author feedback, and they were split on the idea of reducing submission page limits. All other proposals received majority support, with the strongest support indicated for strict penalties for under-declared conflicts.

On the topic of aiding conflict declaration, we organized options in a Matrix table with the Likert scale: not useful, somewhat useful, very useful. Participants indicated a strong preference for automated COI entry (Figure 5). Systems like CLOSET [1] can provide some support for this function, but challenges with false positives and entity resolution issues remain. Ultimately,

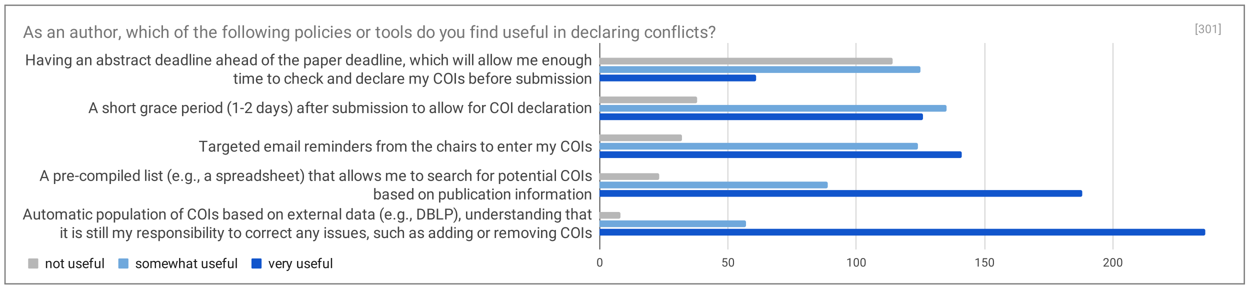


Figure 5: Respondents showed a clear preference for automated methods for COI entry.

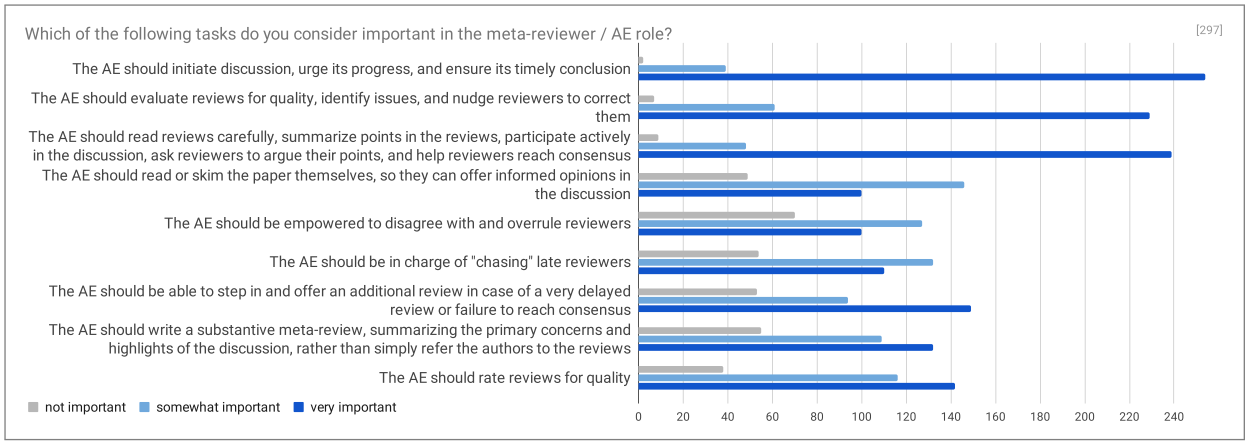


Figure 6: A significant majority of participants consider all of the above AE expectations somewhat or very important.

some COI information is not in the public domain, but the community's clear desire for such support perhaps indicates that we should consider possible initiatives in this direction. Out of the 301 respondents, only 61 considered an abstract deadline a very useful buffer for taking care of their COI entry, whereas having a short grace period after the regular deadline was more popular (42% consider it very useful and 45% somewhat useful).

2.4 AE Expectations

With the growth of our research community, many of our PCs have grown larger to handle the increasing number of submissions. This has led to a hierarchical approach in PC organization, with a small set of PC members acting in the role of Area Chairs, Associate Editors, or meta-reviewers—for brevity, we will refer to all such members as AEs from here on. Frequently, AEs do not review papers directly but are responsible for handling a set of submissions, coordinating discussions, identifying issues with the reviews and taking appropriate action, etc.

With a large number of submissions, PC-chairs are often unable to keep a close eye on reviews and discussions of all submissions, so the function of AEs is critical. However, we observe significant variability in AEs' involvement and submission handling, despite PC-chairs often sharing expectations through guideline documents. We wanted to better understand, through the

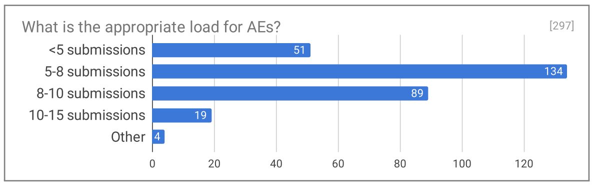


Figure 7: Participants were asked to suggest an appropriate AE load in the context of a single submission cycle with about 200 submissions.

survey, our community's expectations of the AE role. We posed to participants the question "Which of the following tasks do you consider important in the expectations for this role?" We organized AE actions in a Matrix table with the Likert scale: not important, somewhat important, very important. Figure 6 shows the aggregate responses of 297 participants.

Based on the results, the most important functions of AEs include initiating and ensuring the progress of discussions, evaluating reviews for quality and asking for corrections, reading the reviews carefully and actively participating in the discussion themselves, urging the reviewers to support their positions with reasoned arguments. We note, however, that all actions are recognized by the majority of participants (76% or more) as somewhat or very important. The item that received the least support, with 24% of participants noting it as not important, was empowering the AEs to disagree with and overrule reviewers. However, we should highlight that

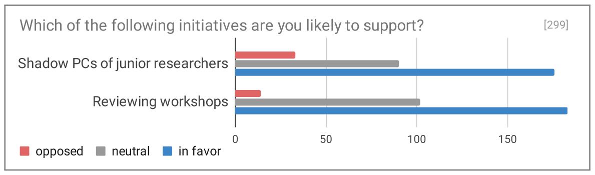


Figure 8: Most participants support these particular initiatives for fostering a more positive reviewing culture, but there is some non-negligible opposition.

even in this case, the overwhelming majority (76%) recognized it as somewhat or very important.

In the context of AE responsibilities, we asked our survey participants to indicate the number of papers that they consider a reasonable load for this role. We recognize that this expectation can be affected by how this load is distributed throughout the year. To alleviate this ambiguity, we asked participants to consider a single submission cycle with about 200 submissions in total, when most reviewers are assigned 3–4 papers to review. (To help the reader put this in context, this is close to the number of submissions typically received in the July cycle of SIGMOD). Out of the 297 participants who answered this question, 45% recommend a load of 5–8 submissions, and 30% recommend a load of 8–10 submissions (Figure 7).

2.5 Fostering Better Culture

We often observe that good and reliable reviewers do a consistently good job, regardless of incentives. Ideally, we want our community to have an established culture of conscientious reviewing. However, PhD students are not consistently trained to be good and conscientious reviewers.[1] Our conferences can establish efforts that support such training, that both hone the reviewing skills of participating researchers and promote good reviewing practices as something valued in our community.

We proposed two potential initiatives that our organizing committees could undertake, and we asked participants' opinions on whether they were likely to support them. The Likert scale we used was: opposed, neutral, in favor. We describe the information we gave to participants about these initiatives below:

Shadow PCs of junior researchers. Junior researchers are added to a shadow PC, and are assigned to papers similar to the regular PC. They can remain anonymous to the AE and to other reviewers. Their reviews are assessed for quality by the AE, and top shadow reviewers are recognized with awards. High-quality shadow reviews can be made available to the authors.

Reviewing workshops. This can be an event collocated with our conferences, where junior researchers are exposed to good and bad reviewing practices through anonymized example reviews. They take a stab at reviewing a mock submission, and senior researchers help them work on their reviews. Successful completion of the workshop earns participants a certificate, which may help them get on PCs sooner.

We received responses from 299 participants, the majority of whom were in favor of both proposals (Figure 8). However, there is some non-negligible opposition as well (about 11% for the shadow PC). A subsequent free-form question in our survey provided some clarity to this stance, as some participants suggested that assigning junior people to the shadow PC may devalue their abilities and contributions, and they would rather allow people to jump into the regular PC directly. We believe that it is meaningful to consider these concerns and adjust and clarify the proposals accordingly. For example, the primary target of the shadow PC may be somewhat junior graduate students who want to gain experience in reviewing but who would not be normally invited to regular program committees at this early stage of their careers.

2.6 General Feedback

Our survey concluded with a free-form textbox, where participants were invited to share further thoughts. We were delighted with the engagement of our community on this topic. We received 150 responses in the suggestions textbox, totaling more than 16K words. Many responses thanked the task force for the initiative, and the vast majority shared thoughts on the challenges and provided interesting suggestions. We found many opposing views expressed in the feedback. For example, many participants urge more detail and clarity in the meta-reviews submitted by AEs, and others express opposition, arguing that meta-reviews sometimes increase confusion when concerns and requests do not match well with those of the reviewers. Another example of conflicting opinions relates to the opportunities for revisions and feedback; many participants urge the community to maintain these functions, and others argue that these efforts have little benefit and only add to the workload of reviewers and authors. We also saw a lot of input on reviewing incentives and penalties, and COI handling. There are also several novel suggestions on releasing reviews of accepted papers (anonymously), and suggestions for empowering reviewers to champion papers.

Given the extent of the feedback we received in this part of the survey, we intend to summarize and discuss more thoroughly these suggestions in a separate report. Our task force is working on producing a list of possible initiatives, with discussion of the potential benefits, drawbacks, and challenges in implementing each one.

[1]We fully recognize that we can find many bad reviewers amongst senior researchers as well. However, on the topic of fostering a culture of conscientious reviewing, it is more practical to reach out to our more junior members.

Requiring submitting authors to serve as reviewers. Every submitting author will be added to the pool of reviewers and should contribute to the peer review process.

Pros: + Increases the pool of reviewers significantly, resulting in reduced load all around.

Cons: – Hard to vet reviewing strengths and expertise of authors, potentially hurting review quality.
– Can make us more prone to unethical behaviors.
– Will make conflict declaration harder, as authors would now have to declare conflicts with all other authors (not just with the PC).

Punitive deterrents for not meeting reviewing expectations. Reviewers who do not meet expectations (e.g., are late in submitting reviews, do not engage in discussions) may be added to blacklists that are shared with other PC chairs or are banned from participating in PCs for some time.

Pros: + May improve review quality and timeliness.

Cons: – Effects of quality are unclear. We would likely need some ground work to establish effective incentives or deterrents.
– Punitive deterrents may disincentivize people from signing on to PCs.

Amplify recognition of good reviewers. Currently, good reviewers are recognized with awards. Perhaps we should amplify these incentives, with more significant recognitions awarded for consistent reviewing contributions across venues, over a period of time. Alternatively, consider financial incentives, e.g., reduced registration for PC members who perform their duties diligently.

Pros: + May improve review quality and timeliness.

Cons: – It is unclear if conference budgets can afford financial incentives.
– It is unclear whether existing reviewer awards have had much impact in reviewing quality.

Reduce the number of reviews per paper. Assign papers to 2 reviewers at first. Papers with 2 rejects are rejected. Papers with at least one borderline or positive review get one more reviewer assigned.

Pros: + Reduces the number of papers that get 3 reviews, thus reducing the load per reviewer.

Cons: – Might reduce acceptance rates

Restricting reviewers from participating in multiple PCs in a given year. There is currently no formal coordination across our venues. The same people may be invited to several committees, resulting in unmanageable reviewing load. Junior members of the community may be more prone to overcommitting.

Pros: + Will reduce reviewer load across our venues, hopefully translating to improved review quality and engagement.

Cons: – May be hard to implement in practice, as it requires coordination across venues.
– If the effort is simply reduced to a recommendation included in the invitation letter, it is unlikely to be heeded.

Pacing reviewing deadlines. Follow the example of conferences in other areas (like Software Engineering), where there are two different deadlines for the reviews. Half (or $\frac{1}{3}$) of the reviews must be delivered on the first deadline, and the rest can be delivered on the second deadline. Usually the first deadline is defined in the middle of the review period.

Pros: + Allows PC chairs to identify early PC members that will potentially deliver reviews late, and act sooner.

De-anonymizing reviews. Move our review processes towards an open science paradigm, where reviews are eponymous and public (e.g., through a system like OpenReview).

Pros: + More accountability in reviewing.
+ Public reviews would incentivize reviewers to do a conscientious job.
+ May shield from bad actors, and may make investigation of ethics issues easier.

Cons: – Possibly harmful to diversity and to junior researchers, who may be concerned to engage in public criticisms.
– Fear of possible retaliation

Increase the number of reviews per paper. By assigning a submission to more than the standard 3 reviewers, we are more likely to get enough reviews on time, and tolerate occasional low-quality reviews.

Pros: + Reduces the stress of chasing late reviewers
+ More likely to get sufficient expert and high quality reviews

Cons: – It increases overall reviewer load

Limiting submissions per author. Our conferences could limit the total number of submissions by each author during the span of a year or of a reviewing cycle.

Pros: + May reduce load by restricting the number of submissions.
+ May curb “paper mills” and low-novelty submissions that clog our systems.

Cons: – Hard to determine the proper cutoff.

Sharing prior reviews when resubmitting to a new venue. Rejected papers from one venue may get resubmitted to another, sometimes with few or no changes. The submission may end up with the same reviewers who are disappointed to see their feedback having been ignored, likely leading to another rejection; this wastes reviewing cycles. Possible solution: Require authors to share previous reviews and explain how they addressed them.

Pros: + Encourages authors to implement changes and address reviewer comments even when the work is submitted elsewhere.
+ Makes more effective use of reviewing efforts.

Cons: – Potential extra work for authors.

Restricting opportunities for revisions and author feedback. The combination of multiple deadlines per year, author feedback phase, and revisions, have resulted in constant reviewing work for PCs.Very few of our papers get direct acceptances. With most submissions going through revision, we impose more work on authors and reviewers. What if we do away with some established policies, such as author feedback or revision cycles? Or, as a middle ground, narrow the criteria for revisions.

Pros: + Reduces reviewing load.
+ Reduces author load with more papers getting direct accepts.

Cons: – May reduce overall acceptance rates.

Strict penalties, such as desk-rejection, for under-declared conflicts. Despite efforts to improve accountability in conflict declaration, such as grace periods for COI entry, targeted reminders to authors, etc., many authors continue to neglect this critical task. Should we impose stricter penalties for such omissions?

Reduce submission page limit. Shorter papers would take less time to review, thus reducing reviewer load. May require a shift in expectations for contributions.

Figure 9: Additional information on reviewing (top) and submission (bottom) policy proposals that participants were asked to evaluate; they could access this information by hovering over the information icon ⓘ next to each option.

We will work to incorporate the community feedback into this list of suggestions, and we will consider ways of releasing some more detailed comments, as long we do not compromise participants' anonymity.

3. SUMMARY AND OUTLOOK

We are very happy with the community engagement with the survey, and we found many of the suggestions inspiring. Specifically, the survey has helped us better understand intricacies of these challenges that we had not appreciated earlier, and gave us inspiration for exploring more ideas. Our task force will work on making progress towards specific and more thoroughly analyzed proposals that we will share in a subsequent report. In the meantime, we invite anyone to contact the task force chairs with any thoughts or requests.

4. REFERENCES

[1] Sourav S. Bhowmick. CLOSET: Data-driven COI detection and management in peer-review venues. *Commun. ACM*, 66(7):70–71, jun 2023.

[2] Nachum Dershowitz and Rakesh M. Verma. Rebutting rebuttals. *Commun. ACM*, 66(9):35–41, aug 2023.

[3] Michael L. Littman. Collusion rings threaten the integrity of computer science research. *Commun. ACM*, 64(6):43–44, may 2021.