SIGMOD Officers, Committees, and Awardees

Chair
Juliana Freire
Computer Science & Engineering
New York University
Brooklyn, New York
USA
+1 646 997 4128
juliana.freire <at> nyu.edu

Vice-Chair
Ihab Francis Ilyas
Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario
CANADA
+1 519 888 4567 ext. 33145
ilyas <at> uwaterloo.ca

Secretary/Treasurer
Fatma Ozcan
IBM Research
Almaden Research Center
San Jose, California
USA
+1 408 927 2737
fozcan <at> us.ibm.com

SIGMOD Executive Committee:
Juliana Freire (Chair), Ihab Francis Ilyas (Vice-Chair), Fatma Ozcan (Treasurer), K. Selçuk Candan, Yanlei Diao, Curtis Dyreson, Christian S. Jensen, Donald Kossmann, and Dan Suciu.

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

SIGMOD Information Director:
Curtis Dyreson, Utah State University

Associate Information Directors:
Huiping Cao, Manfred Jeusfeld, Asterios Katsifodimos, Georgia Koutrika, Wim Martens

SIGMOD Record Editor-in-Chief:
Yanlei Diao, University of Massachusetts Amherst

SIGMOD Record Associate Editors:
Vanessa Braganholo, Marco Brambilla, Chee Yong Chan, Rada Chirkova, Zachary Ives, Anastasios Kementsietsidis, Jeffrey Naughton, Frank Neven, Olga Papaemmanouil, Aditya Parameswaran, Alkis Simitsis, Wang-Chiew Tan, Pinar Tözün, Marianne Winslett, and Jun Yang

SIGMOD Conference Coordinator:
K. Selçuk Candan, Arizona State University

PODS Executive Committee:
Dan Suciu (Chair), Tova Milo, Diego Calvanse, Wang-Chiew Tan, Rick Hull, Floris Geerts

Sister Society Liaisons:
Raghu Ramakhrishnan (SIGKDD), Yannis Ioannidis (EDBT Endowment), Christian Jensen (IEEE TKDE).

Awards Committee:
Martin Kersten (Chair), Surajit Chadhuri, David DeWitt, Sunita Sarawagi, Mike Carey

Jim Gray Doctoral Dissertation Award Committee:
Ioana Manolescu (co-Chair), Lucian Popa (co-Chair), Peter Bailis, Michael Cafarella, Feifei Li, Qiong Luo, Felix Naumann, Pinar Tozun

SIGMOD Systems Award Committee:
Mike Stonebraker (Chair), Make Cafarella, Mike Carey, Yanlei Diao, Paul Larson

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)
David DeWitt (1995)
Serge Abiteboul (1998)
Rudolf Bayer (2001)
Ronald Fagin (2004)
Jennifer Widom (2007)
Umeshwar Dayal (2010)
Stefano Ceri (2013)
Gerhard Weikum (2016)
Jim Gray (1993)
C. Mohan (1996)
Hector Garcia-Molina (1999)
Patricia Selinger (2002)
Michael Carey (2005)
Moshe Y. Vardi (2008)
Surajit Chaudhuri (2011)
Martin Kersten (2014)
Goetz Graefe (2017)
Philip Bernstein (1994)
David Maier (1997)
Rakesh Agrawal (2000)
Don Chamberlin (2003)
Jeffrey D. Ullman (2006)
Masaru Kitsuregawa (2009)
Bruce Lindsay (2012)
Laura Haas (2015)
Raghu Ramakrishnan (2018)

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, Utkarsh Srivastava (2018)

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)
Jeffrey Ullman (1996)
Raghu Ramakrishnan (1999)
Daniel Rosenkrantz (2001)
Surajit Chaudhuri (2004)
Hans-Jörg Schek (2007)
David Lomet (2010)
H.V. Jagadish (2013)
Samuel Madden (2016)
Gio Wiederhold (1995)
Avi Silberschatz (1997)
Michael Carey (2000)
Richard Snodgrass (2002)
Hongjun Lu (2005)
Klaus R. Dittrich (2008)
Gerhard Weikum (2011)
Kyu-Young Whang (2014)
Yannis E. Ioannidis (2017)
Yahiko Kambayashi (1995)
Won Kim (1998)
Laura Haas (2000)
Michael Ley (2003)
Tamer Özsu (2006)
Beng Chin Ooi (2009)
Marianne Winslett (2012)
Curtis Dyreson (2015)
Z. Meral Özsoyoğlu (2018)

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

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

[Last updated : June 30, 2018]

Editor's Notes

Welcome to the December 2018 issue of the ACM SIGMOD Record!

This issue starts with the Database Principles column featuring an article on graph queries by Bonifati and Dumbrava. The article provides an overview of topics in graph processing, focusing on graph-query language fragments that are both tractable from the complexity standpoint and practically relevant. The authors summarize complexity results for graph-query evaluation and containment for a variety of language fragments, and outline foundations for an evaluation and incremental view maintenance engine for one such fragment, regular queries. Other topics discussed in the column include approaches to evaluating complex regular path queries, namely approximate query evaluation and path query learning. The column also outlines challenges encountered in the work on query benchmarking and log analysis, thus inviting the readers to participate in new lines of research.

The Distinguished Profiles column features Andrew Chien, Professor at the University of Chicago, previously vice president/director of Intel Research. Andrew is an ACM Fellow, an IEEE Fellow, and a Fellow of the American Association for the Advancement of Science. In this interview, Andrew shares his experience and vision concerning a range of challenges in emerging and future computational systems. He notes that the SIGMOD community is uniquely positioned to address such challenges, because its researchers have thought of computation and data in an integrated form. He then outlines major trends and directions that SIGMOD researchers may be interested in. Andrew also gives advice for fledging and mid-career database researchers, and shares his desire to explore and experiment with the exciting new systems produced by the community and industry.

The Centers column features an article by Abedjan, Breß, Markl, Rabl, and Soto on data-management systems research at TU Berlin, Germany. The research is advanced with several groups and encompasses a wide range of topics discussed in detail in the article, including scalable data processing, modern hardware, benchmarking, and data integration. The authors also discuss funding support for the group, research translation into products and systems, and ways in which the research group has supported the database community.

The Reports Column features two articles. The first article is the SIGMOD 2018 Program Committee Chair's Report by Philip Bernstein. The report provides detailed information about the submissions, the reviewing process, decisions on revisions and acceptance, and the conference program, including notes on changes from previous years. Bernstein also shares observations and thoughts on the reviewing quality, and provides detailed recommendations on producing good reviews. Historical notes in the article cover Bernstein's experience and contributions as PC Chair for SIGMOD 1979 and VLDB 2002. The article concludes with a remark on how the conference reviewing process could be improved, and with acknowledgments to the SIGMOD 2018 PC members and vice chairs. The second article in the Reports Column reports on the proceedings of the First International Workshop on Incremental Recomputation: Provenance and Beyond (IRPb). The goal of the workshop was to explore the breadth and depth of the recomputation problem, with specific emphasis on the role of provenance in the area. The workshop topics were characterized using the categories of recomputation, incremental computation, approximate computation, and provenance. The contributions also covered a variety of application domains. The report summarizes the contributions and provides a pointer to the abstracts and presentations.

On behalf of the SIGMOD Record Editorial board, I hope that you enjoy reading the December 2018 issue of the SIGMOD Record!

Your submissions to the SIGMOD Record are welcome via the submission site:
http://sigmod.hosting.acm.org/record

Prior to submission, please read the Editorial Policy on the SIGMOD Record's website:
https://sigmodrecord.org

Yanlei Diao and Rada Chirkova

December 2018

Past SIGMOD Record Editors:

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

Graph Queries: From Theory to Practice

Angela Bonifati
University of Lyon 1 & CNRS LIRIS
{angela.bonifati}@univ-lyon1.fr

Stefania Dumbrava*
ENS Rennes & CNRS IRISA & INRIA
{stefania-gabriela.dumbrava}@ens-rennes.fr

ABSTRACT

We review various graph query language fragments that are both theoretically tractable and practically relevant. We focus on the most expressive one that retains these properties and use it as a stepping stone to examine the underpinnings of graph query evaluation along graph view maintenance. Further broadening the scope of the discussion, we then consider alternative processing techniques for graph queries, based on graph summarization and path query learning. We conclude by pinpointing the open research directions in this emerging area.

1. INTRODUCTION

Graphs are semantically rich data models able to represent inherently complex object structures and their interconnectivity relationships. Due to their high expressivity, graphs are used in numerous domains, including Knowledge Representation and the Semantic Web, Linked Open Data, geolocation data, as well as life science repositories, such as those used in medicine, biology, and chemistry. Several graph datasets, such as DBPedia [11], Wikidata [40], and Bio2RDF [34], to name a few, are readily available and exhibit a continuous growth, as new user content is injected on a daily basis. Hence, such massive, graph-shaped data have tremendous potential to be queried and explored for knowledge extraction purposes [17].

In this paper, which summarizes our previous work in this area [16, 13, 23], we survey established theoretical graph query foundations and discuss their practical impact. To this end, we examine graph query evaluation and incremental maintenance techniques, along with implementation aspects. We also review alternative graph query processing techniques, such as approximate query evaluation and path query learning.

*Work mainly done while affiliated with University of Lyon 1 & CNRS LIRIS.

As opposed to their relational counterparts, graph queries are recursive in nature and need to inspect both the structure and the heterogeneity of the underlying data. We illustrate this aspect with the following user-specified query that we have taken from the online Wikidata query set, formulated by real users at the Wikidata SPARQL query service [1]. The query outputs the geolocation information of all hospitals in the Wikipedia ontology at a world-wide scale:

```
Q1: SELECT * WHERE {
        ?item wdt:P31*/wdt:P279* wd:Q16917;
              wdt:P625 ?geo .
}
```

Note that wd:Q16917 is a hospital item, wdt:P31 and wdt:P279 are the "instance of" and "subclass of" Wikidata properties, while wdt:P625 is a coordinate location property. Such a query relies on a recursive expression of the kind a^*/b^*, which drives the navigation of the Wikipedia ontology to find all possible occurrences of hospitals, as item instances or subclasses. More precisely, the above query retrieves a set of geolocation data points that represent the positions of hospitals in a map. As such, its result represents *a graph* of interconnected hospital locations. Concerning the language fragment to which this query belongs, we can classify it as a Conjunctive Regular Path query, belonging to C2RPQ, a notable query fragment that we discuss in Section 2. Due to the presence of recursion, such a query performs complex navigation on the Wikidata graph. An alternative example of a graph query is the following Wikidata one, which retrieves a single aggregate value, namely the total number of humans in Wikidata [2].

```
Q2: SELECT (COUNT(?item) AS ?count)
WHERE {
        ?item wdt:P31/wdt:P279* wd:Q5 .
}
```

[1] https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples

[2] Amounting to 4531233 (on September 25, 2018).

In such a case, a property path of the type a/b^* allows to navigate the Wikidata ontology.

While the first query belongs to C2RPQ, the second query is a counting regular path query (a fragment henceforth named RPQ_C). Even though these fragments have remarkable differences in terms of their evaluation complexity (see Section 3), they are also significantly different in terms of their retrieved output. In fact, while the result of the first query is a geographical network, the second query retrieves a *semantically rich aggregate value*.

This wide array of possibilities, in terms of query input and output, along with the complexity of query evaluation, drives and motivates our current research in the area [23, 16, 22, 18, 8, 15, 7, 13, 12, 9], part of which we revisit in this paper.

We start by describing the *actual* expressivity and computational complexity of practical graph query fragments, as used in various modern graph query languages [3], and focus on how to efficiently process them. We also expand on ensuring the reliability of potentially security-critical applications that can leverage queries in the above languages. To this end, we illustrate in [16] the feasibility of employing formal methods to formalize the expressive regular query (RQ) language and to mechanize the implementation of a corresponding *inference and view maintenance* engine, whose correct behavior we establish through machine-checked proofs.

Hence, we turn to RPQs and study approximate query processing (AQP), which gives the users the agency to decide the tradeoff w.r.t efficiency for query fragments that are expensive to process. In particular, in [23], we investigate the effectiveness of storing pre-computed aggregates to approximate the result of RPQ_C queries, which have a high runtime evaluation cost. To this purpose, we illustrate a query-driven summarization algorithm that we introduced. As we will outline in Section 4, we tackle reachability preservation, in the presence of aggregates, with the explicit purpose of obtaining a small, *reusable graph summary* that can lend itself easily to in-database approximate evaluation.

Next, we illustrate *path query learning* as an alternative processing technique for graph queries [13]. Still on the RPQ fragment, we show how to infer a query statement from a set of positive and negative examples, the latter embodying the expected (or not) query results. As the consistency checking problem for RPQ queries is PSPACE-complete, we resort to lifting the soundness condition of the learning algorithm and propose a learning algorithm that selects paths of a given length. Finally, we discuss an interactive scenario, which leads to a learning algorithm that starts with an empty sample and continuously interacts with the user, in order to infer the goal query.

The paper is organized as follows. We start in Section 2 with an outline of the underlying graph data models and the fundamental graph query fragments that have been studied in the literature and identified as retaining practical interest. Section 3 expands on the complexity of query and incremental view evaluation and illustrates both evaluations for a highly expressive query fragment. Section 4 describes approximate analytical processing and path query learning. Finally, we conclude in Section 5, by highlighting open problems in this area and by providing future research directions.

2. PRELIMINARIES

Graph Database Models. Graph databases rely on *nodes*, to denote abstract entities, and on *edges*, to denote the relationships between them. Such is the structure of the basic edge-labeled model, which we consider in Section 3. This can be further enhanced, to account for *direction*, by taking edges to be ordered pairs of vertices, for *heterogeneity*, by allowing multiple edges between a given pair of vertices, as well as multiple labels, on both vertices and edges, and for *data storage*, by allowing an arbitrary number of properties, or key/value pairs, to be attached to both vertices and edges. Considering these extensions, we reach the expressivity level of the *property graph model* (PGM) [2, 17], on which we focus in Section 4 and which we define next.

Given a finite sets of symbols (*labels*) Σ, *property keys* $\mathcal{K}$, and *property values* $\mathcal{N}$, a *property graph instance* $\mathcal{G}$ over (Σ, $\mathcal{K}$, $\mathcal{N}$), is a structure $(\mathcal{V}, E, \eta, \lambda, \nu)$, such that $\mathcal{V}$ and E are finite sets of *vertex/edge* identifiers, $\eta : E \to \mathcal{V} \times \mathcal{V}$ associates a pair of vertex identifiers to each edge identifier, $\lambda : \mathcal{V} \cup E \to \mathcal{P}(\Sigma)$ [3] associates a set of labels to vertex/edges, and $\nu : (\mathcal{V} \cup E) \times \mathcal{K} \to \mathcal{N}$, associates a value to each vertex/edge property key.

Example 1. *We exemplify the base, edge-labeled model with the graph instance $\mathcal{G}_{SN}$ in Fig. 1, which represents a social network, whose schema is inspired by the LDBC benchmark [24]. Entities are customers (type* `Person`*, C_i),* connected *(l_0) and/or* following *(l_1) each other, that can purchase (l_4) merchandise (type* `Product`*, M_i). This is* promoted *(l_5) in ads (type* `Message`*, A_i), which are* posted *(l_3) on brand pages (type* `Forum`*, P_i),* moderated *(l_2) by specific persons. Additionally, customers can* endorse *(l_6) each other or* endorse *a brand.*

[3] i.e., $\mathcal{P}(\Sigma)$ denotes the set of finite subsets of Σ.

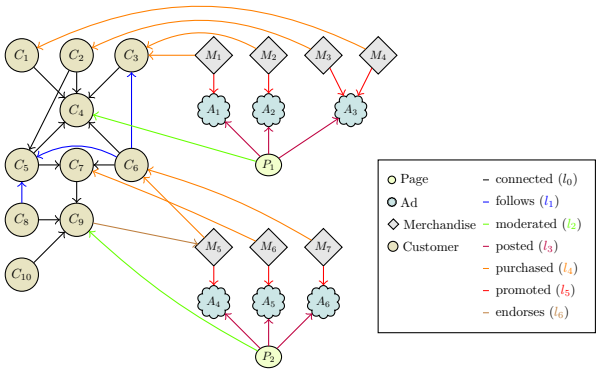


Figure 1: Example Social Graph $\mathcal{G}_{SN}$

Fundamental Query Fragments. *Regular expressions* over a finite alphabet Σ are defined as $e ::= \epsilon \mid s$, with $s \in \Sigma \mid e + e \mid e \cdot e \mid e^*$. A *regular language* $\mathcal{L}(\Sigma)$ of complex labels can thus be inductively built: $\mathcal{L}(\epsilon) = \{\epsilon\}$; $\mathcal{L}(s) = \{s \mid s \in \Sigma\}$; $\mathcal{L}(e_1 + e_2) = \mathcal{L}(e_1) \cup \mathcal{L}(e_2)$; $\mathcal{L}(e_1 \cdot e_2) = \mathcal{L}(e_1) \cdot \mathcal{L}(e_2)$; $\mathcal{L}(e^*) = \{e_1 \cdot e_2 \mid e_1 \in \mathcal{L}(e) \wedge e_2 \in \mathcal{L}(e^*)\}$. Next, given $\mathcal{V}$, a countably infinite set of variables, and $\mathcal{D}$, a domain of constant values, *terms* are elements of $\mathcal{V} \cup \mathcal{D}$. Based on these building blocks, the following prominent *query* fragments have emerged: 1) **regular path queries**, $RPQ = \{s(t_1, t_2) | s \in \mathcal{L}(\Sigma)\}$, 2) **counting** RPQ, $RPQ_C = \{s(t_1, t_2) | s \in \mathcal{L}(\Sigma \cup \{count\})\}$, 3) **2-way** RPQs, which allow backward navigation, $2RPQ = \{s(t_1, t_2) | s \in \mathcal{L}(\Sigma \cup \{s^- | s \in \Sigma\})\}$, 4) **conjunctive** $2RPQ$, $C2RPQ = \{\bigwedge_{i \in \mathbb{N}} s_i(t_1, t_2) | s_i \in 2RPQ\}$, 5) **union of** $C2RPQ$, $UC2RPQ = \{\bigvee_{i \in \mathbb{N}} s_i(t_1, t_2) | s_i \in C2RPQ\}$, 6) **nested** $UC2RPQ$, $nUC2RPQ = \{\bigvee_{i \in \mathbb{N}} s_i(t_1, t_2) | s_i \in \{s^* | s \in UC2RPQ\}\}$, 7) **union of conjunctive, nested** $2RPQ$, $UCN2RPQ = \{\bigvee_{i \in \mathbb{N}} \bigwedge_{j \in \mathbb{N}} s_{i,j}(t_1, t_2) | s_{i,j} \in \{s^* | s \in 2RPQ\}\}$, and the recent 8) **regular queries**, $RQ = \{s(t_1, t_2) | s \in \{s^* | s \in UCN2RPQ\}\}$.

The most expressive graph query fragment, RQ (regular queries) [39, 36], is an extension of unions of conjunctive 2-way regular path queries (UC2RPQs) and of unions of conjunctive nested 2-way regular path queries (UCN2RPQs). Regular queries support expressing complex regular patterns between graph nodes. They also correspond to Datalog with linear recursion [31], also known as non-recursive Datalog, extended, at the language-level, with transitive closures of binary predicates.

EXAMPLE 2. *Revisiting Example 1, we illustrate the above query fragments on the social graph $\mathcal{G}_{SN}$ from Figure 1:*

$$
\begin{aligned}
&Q_1 : \Omega_1(X, \mathcal{B}) \leftarrow (l_0 + l_1)^* \cdot l_6(X, \mathcal{B}) \\
&Q_2 : \Omega_2(X, \mathcal{B}) \leftarrow l_4^- \cdot l_5 \cdot l_3^-(X, \mathcal{B}) \\
&Q_3 : \Omega_3(C, \_) \leftarrow l_4^- \cdot l_5 \cdot l_3^-(X, \mathcal{B}), count(X, C) \\
&Q_4 : \Omega_4(X, \mathcal{B}) \leftarrow l_2^-(X, \mathcal{B}), l_6 \cdot l_5 \cdot l_3^-(X, \mathcal{B}) \\
&Q_5 : \Omega_5(X, Y) \leftarrow l_4 \cdot \Omega_4{}^+ \cdot l_4(X, Y) \\
&Q_6 : \Omega_6(X, Y) \leftarrow l_3 \cdot l_5(\mathcal{B}, X), \Omega_5(X, Y), l_3 \cdot l_5(\mathcal{B}, Y) \\
&Q_7 : \Omega_7(X, Y) \leftarrow (l_6^- \cdot (l_0 + l_1)^+ + \Omega_6)(X, Y) \\
&Q_8 : \Omega_8(X, Y) \leftarrow (l_4^- \cdot \Omega_7 \cdot l_4)^+(X, Y)
\end{aligned}
$$

Consider $\mathcal{B}$ to be a brand in the social graph $\mathcal{G}_{SN}$. Ω_1 returns the customers connected, directly or indirectly, to an endorser*, while Ω_2 returns $\mathcal{B}$'s customers. Ω_3 counts the above, while Ω_4 returns $\mathcal{B}$'s* fans*, i.e., the customers that monitor a $\mathcal{B}$ page and endorse its merchandise. Ω_5 returns the products that are* viral*, i.e., purchased (or endorsed, for Ω_7) by connected brand fans (of brand $\mathcal{B}$, for Ω_6). Ω_8 returns all consumers that purchase viral $\mathcal{B}$ products. In terms of query expressivity, the following memberships hold: $Q_1 \in RPQ, Q_2 \in 2RPQ, Q_3 \in RPQ_c, Q_4 \in C2RPQ, Q_5 \in UC2RPQ, Q_6 \in nUC2RPQ, Q_7 \in UCN2RPQ, Q_8 \in RQ$.*

3. GRAPH QUERY PROCESSING

Our discussion on the evaluation and processing of graph queries begins in Section 3.1, with an brief overview of the respective complexity of the various query classes we discuss. Narrowing down on Regular Queries, which represent the most expressive fragment, we then proceed, in Section 3.2, to presenting the design of a *custom RQ evaluation and incremental maintenance algorithm*. In Section 3.3, we provide insights concerning its corresponding implementation and formal development, carried out with the Coq proof assistant [37]. To the best of our knowledge, this work constitutes the first *certified specification and implementation* of the RQ language and of its mechanized processing engine.

3.1 Complexity Results

In Table 1, we summarize the main complexity results [36, 39] regarding the evaluation and containment of the graph query classes in Section 2. Note that, at a foundational level, these fragments rely on *conjunctive queries* (CQ), whose evaluation is polynomial and whose containment is NP-complete [19]. Their common denominator is that they perform *edge traversals* (through join chains), while specifying and checking the existence of *constrained paths*.

The most expressive class we consider is RQ (*Regular Queries*) [36], already described in Section 2. Unlike full Datalog [1], with P-complete evaluation and undecidable containment, Regular Datalog is particularly well-behaved. First, its evaluation

$$
\begin{array}{rcll}
t & ::= & n \in \mathcal{D} \mid x \in \mathcal{V} & \text{(Term)} \\
A & ::= & s(t_1, t_2), s \in \Sigma \mid t_1 = t_2 & \text{(Atom)} \\
L & ::= & A \mid A^+ & \text{(Literal)} \\
B & ::= & L_1 \wedge \ldots \wedge L_n & \text{(Conj. Body)} \\
D & ::= & B_1 \vee \ldots \vee B_n & \text{(Disj. Body)} \\
C & ::= & (t_1, t_2) \leftarrow D & \text{(Clause)} \\
\Pi & ::= & \Sigma \rightarrow \{C_1, \ldots, C_n\} & \text{(Program)}
\end{array}
$$

Figure 2: Regular Datalog Grammar

has NLOGSPACE-complete data complexity and is hence included in NC, the class of *highly parallelizable* problems. Second, the containment of Regular Datalog queries is *decidable*, with an elementary tight bound (2EXPSPACE-complete) [36].

The behaviors of the other fragments presented in Table 1 resemble each other, with two exceptions. The evaluation of counting label-constrained reachability queries RPQ_C has $\#P-$complete data complexity [38] and its containment problem is undefined. The containment problem for RPQ and 2RPQ is PSPACE-complete [6]. For the sake of conciseness, we omit here further details on the complexity of evaluation of special classes of RPQ, boiling down to the trichotomy in [9] and to simple transitive expressions in [33].

3.2 Evaluation and Maintenance

We present the theoretical foundations of an evaluation and incremental view maintenance engine for regular queries (RQ). We begin with a high-level description of the basic algorithm underpinning evaluation and then extend the introduced constructs to support *incremental maintenance*.

Evaluation. As a subset of Datalog, RQs can consequently lend themselves to the same evaluation techniques employed by deductive reasoning engines. The adopted evaluation strategies of these engines can be classified as: 1) *bottom-up*, i.e., start from the extensional database and generate new facts by forward-chaining, 2) *top-down*, i.e., start from the query (part of the intensional database) and construct a proof tree or a refutation proof by back-chaining, or 3) *rewriting-based*, i.e., transform the query into one for which bottom-up evaluation emulates top-down information passing (*magic-sets* [10]) or pushdown automata (*chain-queries* [28]). Henceforth, we focus on the bottom-up approach, and build on it in order to construct the first inference engine. Our choice is motivated by the desirable properties of bottom-up inference, such as guaranteed termination for finite models and amenability to formalization. Specifically, we rely on its fundamentally set-theoretical nature to specify the RQ engine's behavior and to construct its machine-checked soundness proof in the Coq proof assistant.

To facilitate efficient mechanical reasoning, we represent RQ constructs as illustrated in Fig. 2. Notably, we formalize programs as mappings from indexing symbols to a *single pair* of source-target nodes and to a *normalized* disjunctive body. The normalized form is obtained through a *completion* procedure, uniformizing clause heads and regrouping their respective bodies. For example, the program $s(a,b).\ s(z,y) \leftarrow p(x,y), q^+(z,x)$ is normalized as $s(x,y) \leftarrow (a = x \wedge b = y) \vee (p(z,y) \wedge q^+(x,z))$ and represented by a *function* from s to the head and disjunctive body. Based on this representation, we define an RQ over a graph $\mathcal{G}$ as a *stratified*, Regular Datalog program Π, along with a distinguished query clause, whose head is the top-level *view*. We illustrate this in Example 3, with $l_r(X,Y)$ as RQ.

EXAMPLE 3. *In $\mathcal{G}_{SN}$, let $\mathcal{B}$ be a brand wanting to determine if a customer pair (l_c) is in the same advertising reachability cluster (l_r). We say that $\mathcal{B}$'s advertising reaches a customer X either: 1) directly (l_d), if X endorses (l_6) or purchases (l_4^-) merchandise promoted (l_5) in ads posted (l_3) on the brand's page, or 2) indirectly (l_i), if X is linked via a follower/connection chain to another customer that is under the direct reach of $\mathcal{B}$.*

$$
\begin{array}{lcl}
l_r(X,Y) & \leftarrow & {l_c}^+(X,Y) \\
l_c(X,Y) & \leftarrow & l_i(X,\mathcal{B}), l_i(Y,\mathcal{B}) \\
l_i(X,\mathcal{B}) & \leftarrow & (l_1 + l_0)^+ \cdot l_d(X,\mathcal{B}) \\
l_d(X,\mathcal{B}) & \leftarrow & (l_6 + l_4^-) \cdot l_5 \cdot l_3(X,\mathcal{B})
\end{array}
$$

Figure 3: Advertising Reachability Clusters

The semantics of Regular Datalog programs follows standard term-model definitions. For optimization purposes, we model *interpretations* $\mathcal{G}$ as indexed relations $(\Sigma \times \{\emptyset, +\}) \rightarrow \mathcal{P}(\mathcal{D} \times \mathcal{D})$, which contain labeled graphs and their transitive closure. Given that closures are thus internalized, we also impose that interpretations be well-formed, i.e., that the information stored in $\mathcal{G}(s,+)$ corresponds to the actual transitive closure of $\mathcal{G}(s,\emptyset)$. Hence, we check if, for every node pair $(n_1,n_2) \in \mathcal{G}(s,+)$, there exists a path (vertex sequence) that starts with n_1 and ends with n_2. Hence, a ground literal $s^m(n_1,n_2)$ is satisfied by $\mathcal{G}$ iff $(n_1,n_2) \in \mathcal{G}(s,m)$, with $m \in \{\emptyset,+\}$, the transitive closure marker. Consequently, a clause, with index s and disjunctive body $D \equiv (L_{1,1} \wedge \ldots \wedge L_{1,n}) \vee \ldots \vee (L_{m,1} \wedge \ldots \wedge L_{m,n})$, is satisfied by $\mathcal{G}$, $\mathcal{G} \models_s (t_1,t_2) \leftarrow D$, iff $\forall \eta, \bigvee_{i=1..m}(\bigwedge_{j=1..n} \mathcal{G} \models \eta(L_{i,j})) \Rightarrow \mathcal{G} \models$

Query Fragment	*Evaluation*	*Containment*
RPQ	NLOGSPACE-complete	PSPACE-complete
RPQ_C	$\#P-$complete	*Undefined*
2RPQ	NLOGSPACE-complete	PSPACE-complete
C2RPQ	NLOGSPACE-complete	EXPSPACE-complete
UC2RPQ	NLOGSPACE-complete	EXPSPACE-complete
nUC2RPQ	NLOGSPACE-complete	EXPSPACE-complete
UCN2RPQs	NLOGSPACE-complete	EXPSPACE-complete
RQ	NLOGSPACE-complete	2EXPSPACE-complete

Table 1: Evaluation and containment data complexity for the language fragments studied in this paper.

$\eta(s(t_1, t_2))$, i.e, for all substitutions η, which ground body literals, if an instantiated body disjunct is satisfied, then so is the instantiated head. The latter is indeed a *ground literal*, as we impose the safety condition that all head variables appear in the body. A well-formed interpretation $\mathcal{G}$ is thus a model for a program Π w.r.t Σ iff $\mathcal{G}$ satisfies all clauses indexed by Σ symbols, i.e., $\mathcal{G} \models_\Sigma \Pi$ iff $\forall\ s \in \Sigma, \mathcal{G} \models_s \Pi(s)$.

To compute models, we implement bottom-up RQ evaluation based on the *consequence operator* [1]. This relies on a generic matching algorithm that, for an initial interpretation and a clause construct, computes the set of all satisfying substitutions. For example, given $\mathcal{G}$ and a literal l, the matching function $M^B_\mathcal{G}(l)$ outputs all substitutions σ, such that $\mathcal{G} \models \sigma(l)$. For a clause, $\Pi(s) \equiv (t_1, t_2) \leftarrow \bigvee_{i=1..n} B_i$, it extends to body matching straightforwardly, with $M^B_\mathcal{G}(B_i)$ traversing B_i and collecting the set of substitutions obtained from the individual matching. Substitutions for each disjunctive clause are thus accumulated and the resulting ground heads, newly inferred facts, are added to the interpretation. The consequence operator, encoding nested-loop join, is expressed set-theoretically as $T^{\Pi,s}(\mathcal{G}) \equiv \{\sigma(t_1, t_2) \mid \sigma \in \bigcup_{i=1..n} M^B_\mathcal{G}(B_i)\}$.

Maintenance. Given updates Δ to a base graph $\mathcal{G}$, the above evaluation procedure non-incrementally maintains the top-level Π view, without reusing or adjusting the previously computed maintenance information. This makes it especially inefficient when few nodes are added to a high-cardinality graph.

To remedy this situation, we extend our previous algorithm to take into account information from previously computed models. The key idea is to restrict matching to graph updates in the spirit of incremental view maintenance for relational databases [27]. For example, let V be a materialized view, defined as the path over two base edges, r and s, i.e., $V(X, Y) \leftarrow r(X, Z), s(Z, Y)$. Notice that this path can also be seen as a join between the binary relations r and s on the Z variable, abbreviated as $V = r \bowtie s$. For *base deltas*, Δr and Δs, we can compute the *view delta* as $\Delta V = (\Delta r \bowtie s) \cup (r \bowtie \Delta s) \cup (\Delta r \bowtie \Delta s)$, or, factoring, as $\Delta V = (\Delta r \bowtie s) \cup (r^\nu \bowtie \Delta s)$, with $r^\nu = r \cup \Delta r$. Hence, $\Delta V = \Delta V_1 \cup \Delta V_2$, with ΔV_1 and ΔV_2 computable via the following delta clauses:

$$\delta_1 : \Delta V_1 \leftarrow \Delta r(X, Z), s(Z, Y)$$
$$\delta_2 : \Delta V_2 \leftarrow r^\nu(X, Z), \Delta s(Z, Y)$$

In general, for $V \leftarrow L_1, \ldots, L_n$ and an additive update Δ, we can determine the view delta $\Delta V[\mathcal{G}; \Delta]$ as the set of facts such that $V[\mathcal{G} :+: \Delta] = V[\mathcal{G}] \cup \Delta V[\mathcal{G}; \Delta]$. To this end, we compute the *delta program* $\delta(V) = \{\delta_i \mid i \in [1, n]\}$, where each *delta clause* δ_i is $V \leftarrow L_1, \ldots, L_{i-1}, \Delta L_i, L^\nu_{i+1}, \ldots, L^\nu_n$.

Note that L^ν_j marks that we match L_j against atoms in $\mathcal{G} \cup \Delta G$ with the same symbol as L_j and ΔL_j marks that we match L_j against atoms in $\Delta\mathcal{G}$ with the same symbol as L_j. We revisit the schema in Example 1, on a slightly different graph instance, to illustrate this *incremental view computation*.

Example 4. *Consider Figure 4a, in which entity Y is* monitored *(l_m) by X, if X is its connection/follower/moderator, and* auto-referrals *(l_{ar}) are computed as cyclic endorsements.*

$$\begin{array}{lcl} l_m(X, Y) & \leftarrow & (l_0 + l_1 + l_2)(X, Y) \\ l_{ar}(X, Y) & \leftarrow & l_6(X, Y), l_6(Y, X) \end{array}$$

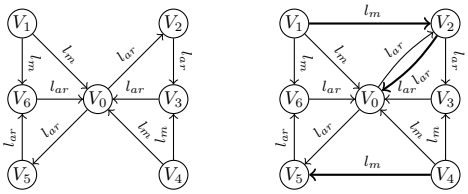
All detectable auto-referrals *are computable with RQ below, as* $[\![\Omega]\!]_\mathcal{G} = \{(V_6, V_0), (V_3, V_0))\}$

$$\Omega(X, Y) \leftarrow l_{ar}(X, Y), l_m(Z, X), l_m(Z, Y)$$

When updating *the previous graph in Figure 4b:* $[\![\Omega]\!]_{\mathcal{G}'} = \{(V_6, V_0), (V_3, V_0), (\mathbf{V_0}, \mathbf{V_2}), (\mathbf{V_2}, \mathbf{V_0}), (\mathbf{V_0}, \mathbf{V_5})\}$. *The delta update* $\Delta\Omega = \{(\mathbf{V_0}, \mathbf{V_2}), (\mathbf{V_2}, \mathbf{V_0}), (\mathbf{V_0}, \mathbf{V_5})\}$ *can be* incrementally *computed from* $\Pi_\Delta = \delta_1 \cup \delta_2 \cup \delta_3$, *as:* $\delta_1 = \emptyset$, $\delta_2 = \{(\mathbf{V_2}, \mathbf{V_0})\}$, $\delta_3 = \{(\mathbf{V_0}, \mathbf{V_2}), (\mathbf{V_0}, \mathbf{V_5})\}$.

3.3 Implementation and Certification

The implementation of the engine accounts for two modes of evaluation: *base* and *incremental*. Note that the former is still needed as, in some cases we identify, incremental evaluation is either not possible or not sensible, as full recomputation may be faster. The built-in engine heuristic is bottom-up and we leverage the non-recursive, stratified nature



(a) Initial Graph (b) Updated Graph

$$
\begin{array}{lrcl}
\delta_1: & \Delta\Omega(X,Y) & \leftarrow & \Delta\mathbf{l_{ar}}(X,Y), \mathbf{l_m}(Z,X), \mathbf{l_m}(Z,Y) \\
\delta_2: & \Delta\Omega(X,Y) & \leftarrow & \mathbf{l_{ar}}^{\nu}(X,Y), \Delta\mathbf{l_m}(Z,X), \mathbf{l_m}(Z,Y) \\
\delta_3: & \Delta\Omega(X,Y) & \leftarrow & \mathbf{l_{ar}}^{\nu}(X,Y), \mathbf{l_m}^{\nu}(Z,X), \Delta\mathbf{l_m}(Z,Y)
\end{array}
$$

(c) Delta Program for Detectable Auto-Referrals

Figure 4: Detectable Auto-Referrals

of the input programs to achieve single-pass, *fine-grained incremental* model computation. In the following, we outline the top-level engine interface and the main *soundness* theorem we formally prove.

Implementation. The *static* parameters of the engine are: a program Π, a graph $\mathcal{G}$, and a symbol set, or *support* supp, indicating the validity of a $\mathcal{G}$ subset, i.e., what information the incremental engine needs not recompute. Indeed, as we will see, a precondition of the engine is that the input graph is a model of Π *up to* supp, i.e., that $\mathcal{G} \models_{\mathsf{supp}} \Pi$. Note that in the database literature, the set of $\mathcal{G}$ symbols, $\mathbf{\Sigma}$, is often seen as a disjoint set pair, $(\mathbf{\Sigma}_E, \mathbf{\Sigma}_I)$, corresponding to the *extensional* and *intensional* program parts. For our engine, this distinction is "dynamic", as the already-processed strata-level is "extensional", or immutable, for the rest of the execution. Thus, typical cases for supp are $\mathsf{supp} \equiv \mathbf{\Sigma}_E$, when the engine has never been run before, or $\mathsf{supp} \equiv \mathbf{\Sigma}$, where $\mathcal{G}$ is the output of a previous run, and thus the consequences for all clauses have been computed. The *dynamic* parameters, capturing the current execution state, are: Δ, the current update, modified at each call, and the already and to-be processed strata, $\Sigma_{\triangleright}$ and $\Sigma_{\triangleleft}$.

Relying on an *incrementality-aware consequence operator*, $T^{\Pi,s}_{\mathcal{G},\mathsf{supp}}(\Delta)$, the engine iterates over $\Sigma_{\triangleleft}$ and, for each unprocessed symbol s, computes its corresponding closure. The algorithm then calls itself recursively, adding both s and s^+ to $\Sigma_{\triangleright}$.

Before discussing the implementation of $T^{\Pi,s}_{\mathcal{G},\mathsf{supp}}(\Delta)$, we explain the modifications made to *base* matching, in order to accommodate delta clauses and programs. Specifically, for each body to be processed incrementally, we generate a *mask*, B_Δ, by marking each of its literals with $m \in \{\mathbf{B}, \mathbf{D}, \mathbf{F}\}$. This indicates whether the engine should match against the base interpretation, the update, or both. We then define incremental atom matching as:

$$
\begin{aligned}
M^{A,m}_{\mathcal{G},\Delta}(a) \quad &= (\text{if } m \in \{\mathbf{B},\mathbf{F}\} \text{ then } M^A_{\mathcal{G}}(a) \text{ else } \emptyset) \cup \\
&\quad (\text{if } m \in \{\mathbf{D},\mathbf{F}\} \text{ then } M^A_{\Delta}(a) \text{ else } \emptyset)
\end{aligned}
$$

Incremental body matching, $M^B_{\mathcal{G},\Delta}$, proceeds as in Section 3.2, but additionally takes into account B_Δ, generated following the *diagonal factoring* below, where each row corresponds to a mask element:

$$
\begin{bmatrix}
{L_1}^{\mathbf{D}} & {L_2}^{\mathbf{F}} & \dots & {L_{n-1}}^{\mathbf{F}} & {L_n}^{\mathbf{F}} \\
{L_1}^{\mathbf{B}} & {L_2}^{\mathbf{D}} & \dots & {L_{n-1}}^{\mathbf{F}} & {L_n}^{\mathbf{F}} \\
\dots & \dots & \dots & \dots & \dots \\
{L_1}^{\mathbf{B}} & {L_2}^{\mathbf{B}} & \dots & {L_{n-1}}^{\mathbf{B}} & {L_n}^{\mathbf{D}}
\end{bmatrix}
$$

Finally, the last piece to complete the incremental engine is the top-level clausal maintenance operator $T^{\Pi,s}_{\mathcal{G},\mathsf{supp}}(\Delta)$ itself. This is more complex than its *base* counterpart, as it must take into account which incrementality heuristics to apply, distinguishing between two cases. If $s \notin \mathsf{supp}$, or Δ contains deletions for any of the literals in the body of $\Pi(s)$, it uses the base operator $T^{\Pi,s}(\mathcal{G} :+: \Delta)$, as we either cannot reuse the previous model or cannot support deletions through our incremental strategy. Otherwise, it generates a body mask, B_Δ, for each of the bodies B, and returns $\bigcup_{B_m \in B_\Delta} M^B_{\mathcal{G},\Delta}(B_m)$.

Certification. Before stating the key result with regard to the correct behavior of our engine, we mention the pre-conditions imposed. First, we require our input programs Π be *stratified*, i.e., that none of its head symbols depend on other that have not been previously defined. Second, as we reason about satisfaction *up to* a given symbol set, Σ, we say that Π is a well-formed slice of Σ, if, for every s in Σ, the symbols defining s in Π are contained in Σ. We establish that the engine operates over well-formed slices, which allows us to isolate reasoning about the current iteration. Finally, we formally prove that the incremental graph view maintenance engine is sound, as stated below.

THEOREM 1. *Let* Π *be a* safe, stratifiable, *Regular Datalog program;* Σ*, its symbols;* $\mathcal{G}$*, a graph instance;* Δ*, an update. The incremental view maintenance engine cumulatively processes each strata symbol, such that, if the already processed symbols,* $\Sigma_{\triangleright}$*, are a well-formed slice, if* Δ *only modifies* $\Sigma_{\triangleright}$*, and if the updated graph is a model of* Π *under* $\Sigma_{\triangleright}$*, then it outputs an* incremental update*, which, when applied to* $\mathcal{G}$*, forms a model of* Π *under* Σ*.*

The proof follows by structural induction on $\Sigma_{\triangleleft}$, relying on results we establish regarding modular satisfaction. These are paramount, as they allow us to reason about satisfaction locally, within each

well-formed slice. Note that the corresponding Coq proof of Theorem 1 is about 25 lines long and, thus, comparable to its paper-version. In total, the library we developed amounts to $\sim 1K$ lines of definitions, specifying our mechanized theory, and ~ 700 lines of proofs. Its compactness is mostly due to the fact that we rely on a library fine-tuned for the computer-aided theorem proving of finite-set theory results. This was built to carry out the mechanized proof of the Feit-Thompson theorem [26] on finite group classification. We leveraged the finite reasoning support, by giving a high-level, mathematical representation of the core engine components, as exemplified with the definition of the consequence operator in Section 3.2. This leads to *composable* lemmas that boil down to set-theoretic statements and, ultimately, to a condensed development, avoiding the proof-complexity explosion characteristic of formal verification efforts.

4. OTHER PROCESSING TECHNIQUES

We present alternative approaches that seek to mitigate the challenges posed by the evaluation of complex RPQ, as discussed in Section 3.1. First, in Section 4.1, we focus on leveraging the expressivity of the property graph model to develop efficient *approximate query evaluation* techniques for the RPQ_C fragment. Second, in Section 4.2, we highlight the promise shown by *path query learning* approaches, in the basic RPQ setting.

4.1 Query Approximation

In the following, we outline a newly introduced algorithm for graph summarization, and its application to the approximate evaluation of RPQ_C queries.

Graph Summarization. Sampling approaches, typically used for approximating relational queries, are not directly applicable to graph processing, due to the lack of the linearity assumption in graph-oriented data [30]. Indeed, the linear relationship between the sample size and execution time typical of relational query processing falls apart in graph query processing. For this reason, we focus on query-driven graph summarization as a baseline technique for untangling approximate graph query processing.

Our effort targets the *efficient*, *high-accuracy*, estimation of RPQ_C analytical queries, known to be costly in terms of runtime. We tackle both challenges, in an effort to achieve an optimal trade-off. First, we seek to obtain a *compact* (yet informative) summary, by explicitly inspecting the query workload and partitioning the graph according to the connectivity of the labels identified as most important. Second, we rely on the expressiveness of the *property graph* model to store *pertinent*, approximation-relevant, data, in the property lists of both nodes and edges. Specifically, these recorded statistics serve the purpose of preserving label-constrained reachability information.

Since both the original and summarized graphs adhere to the property graph data model (as presented in Section 2), the approximate evaluation can be done directly inside the graph database itself, thanks to a seamless query translation we provide.

We now focus on explaining and illustrating the underlying summarization algorithm. Let $\mathcal{G} = (\mathcal{V}, E)$ be a graph with edge labels $\Lambda(\mathcal{G})$. We introduce a summarization algorithm that compresses $\mathcal{G}$ to an AQP-amenable property graph, $\hat{\mathcal{G}}$, tailored for counting label-constrained reachability queries, with labels in Λ_Q, where $\Lambda_Q \subseteq \Lambda(\mathcal{G})$.

The summarization algorithm consists of the following three phases. First, the grouping phase computes Φ, a label-driven partitioning of $\mathcal{G}$ into *groupings*, following the label connectivity on the most frequent labels in $\Lambda(\mathcal{G})$. Next, the evaluation phase refines the previous step, further isolating into *supernodes* the grouping components that satisfy a custom property concerning label-connectivity. The merge phase then coalesces supernodes into *hypernodes*, based on label-reachability similarity conditions, as specified the heuristic mode m.

The *grouping phase* returns a partitioning Φ of $\mathcal{G}$, such that $|\Phi|$ is *minimized* and, for each $\mathcal{G}_i \in \Phi$, the number of occurrences of the most frequent edge label in $\Lambda(\mathcal{G}_i)$, $\max_{l \in \Lambda(\mathcal{G}_i)} (\#l)$, is *maximized*. Hence, we first sort the edge label set $\Lambda(\mathcal{G})$ into a *frequency list*, $\overrightarrow{\Lambda(\mathcal{G})}$. For each $l_i \in \overrightarrow{\Lambda(\mathcal{G})}$, in descending frequency order, we identify the largest $\mathcal{G}$-subgraphs that are weakly-connected on l_i.

EXAMPLE 5. *Let $\mathcal{G}$ be the graph from Figure 1. It holds that: $\#l_0 = 11$, $\#l_1 = 3$, $\#l_2 = 2$, $\#l_3 = 6$, $\#l_4 = 7$, $\#l_5 = 7$, $\#l_6 = 1$. Hence, we can take $\overrightarrow{\Lambda(\mathcal{G})} = [l_0, l_5, l_4, l_3, l_1, l_2, l_6]$. Note that, as $\#l_4 = \#l_5$, we can choose an arbitrary order for the labels in $\overrightarrow{\Lambda(\mathcal{G})}$. We first add $\mathcal{G}_1$ to Φ, as it regroups the maximal weakly-label components on l_0. Hence, $\mathcal{V} = \{R_1 - R_7, M_1 - M_6, F_1, F_2\}$. Next, we add $\mathcal{G}_2$ to Φ, as it regroups the maximally weakly-label component on l_5. We obtain $\mathcal{V} = \{F_1, F_2\}$ and $\Phi = \{\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3\}$, as shown in Figure 5a.*

The *evaluation phase* takes as input Φ, the previously obtained $\mathcal{G}$-partitioning, together with Λ_Q, and outputs $\mathcal{G}^* = (\mathcal{V}^*, E^*)$, an AQP-amenable com-

pression of $\mathcal{G}$. The phase computes $\mathcal{V}^*$, the set of *supernodes* (SN), and E^*, the set of *superedges* (SE). After each step, $\mathcal{G}^*$ is enriched with AQP-relevant properties, such as: $\mathcal{V}$*Weight* and *EWeight*, the number of inner vertices and edges; *LPercent*, the percentage-wise label occurrence; and *LReach*, the number of vertex pairs connected by an edge with a given label. We also record pairwise label-traversal information, such as: *EReach*, the number of paths between two cross-edges with given labels, directions, and common node; and δ, the number of *traversal edges*, i.e., inner/cross-edge pairs, with given labels, directions, and common endpoint. Finally, we compute $\mathcal{V}_F$, the number of *frontier vertices*, given a fixed label and direction, as well as δ, the *relative label participation*, i.e., the number of cross-edges on a given label, relative to that of frontier vertices on another label.

The *merge phase* takes as input the $\mathcal{G}^*$ graph and Λ_Q and outputs a compressed graph, $\hat{\mathcal{G}} = (\hat{\mathcal{V}}, \hat{E})$. The phase proceeds in two steps, corresponding to the creation of $\hat{\mathcal{V}}$, the set of *hypernodes* (HN), and, respectively, to that of $\hat{E}$, the set of *hyperedges* (HE). HNs are computed by merging together supernodes based on two criteria. The primary, *inner-merge*, condition for candidate supernodes requires them to be maximal weakly label-connected on the same label. The *source-merge* heuristic additionally requires that they share the same set of outgoing labels, while the *target-merge* heuristic requires that they share the same set of ingoing labels. HEs are obtained by merging superedges that share the same label and endpoints. Finally, $\hat{\mathcal{G}}$ is enriched with previous AQP-relevant properties, with the addition of $\mathcal{V}^*Weight$, the average SN weight in each HN.

The three phases of the summarization algorithm are illustrated on our running example in Figure 5.

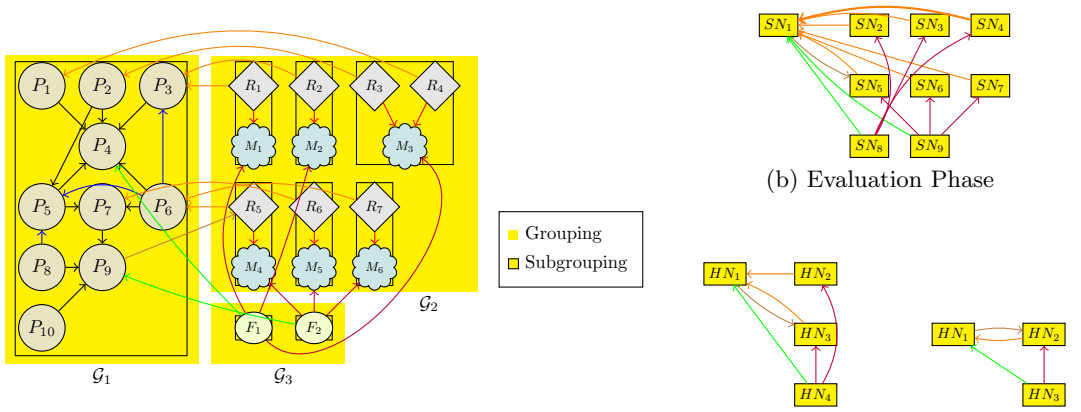
Optimal Summarization: NP-Completeness. We prove the intractability of the optimal graph summarization problem, under the conditions of our algorithm. Specifically, let $\mathcal{G} = (\mathcal{V}, E)$ and $\Phi = \{\mathcal{G}_i = (\mathcal{V}_i, E_i) \mid i \in [1, |\mathcal{V}|]\}$, a $\mathcal{G}$-partitioning. Each HN in $\mathcal{G}_i \in \Phi$ contains HN-subgraphs, $\mathcal{G}_i^k$, that are all *maximal weakly label-connected* on a label $l \in \Lambda(\mathcal{G})$. A *summarization function* $\chi_\Lambda : \mathcal{V} \to \mathbb{N}$ assigns to each vertex, v, a unique HN identifier $\chi_\Lambda(v) \in [1, k]$. χ_Λ is *valid*, if, for any v_1, v_2, where $\chi_\Lambda(v_1) = \chi_\Lambda(v_2)$, v_1, v_2 are in the following cases.
Case 1: part of the same HN-subgraph, $\mathcal{G}_i^k$, that is maximal weak label-connected on l.
Case 2: part of different HN-subgraphs, $\mathcal{G}_i^{k_1}$, $\mathcal{G}_i^{k_2}$, each maximal label-connected on l and not connected by an l-labeled edge in $\mathcal{G}$.

THEOREM 2. *Let* MinSummary *be the problem that, for a graph $\mathcal{G}$ and an integer $k' \geq 2$, decides if there exists a label-driven partitioning Φ of $\mathcal{G}$, $|\Phi| \leq k'$, such that χ_Λ is a* valid summarization. *MinSummary is NP-complete, even for undirected graphs, $|\Lambda(\mathcal{G})| \leq 2$ and $k' = 2$.*

Approximate Query Evaluation. For a graph $\mathcal{G}$ and a counting reachability query Q, we approximate the result $[\![Q]\!]_\mathcal{G}$ of evaluating Q over $\mathcal{G}$. Hence, we translate Q into a query Q^T, evaluated over the summarization $\hat{\mathcal{G}}$ of $\mathcal{G}$, such that $[\![Q^T]\!]_{\hat{\mathcal{G}}} \approx [\![Q]\!]_\mathcal{G}$, as discussed next.
Simple and Optional Label Queries. There are two configurations in which a label l can occur in $\hat{\mathcal{G}}$: either within a HN or on a cross-edge. Thus, we either cumulate the number of l-labeled HN inner-edges or the l-labeled cross-edge weights. To account for the potential absence of l, we also estimate, in the optional-label queries, the number of nodes in $\hat{\mathcal{G}}$, by cumulating those in each HN.
Kleene Plus and Kleene Star Queries. To estimate l^+, we cumulate the counts within HNs containing l-labeled inner-edges and, as above, the weights on l-labeled cross-edges. For the first part, we use the statistics gathered during the *evaluation phase*. We distinguish three scenarios, depending on whether the l_+ reachability is due to: 1) inner-edge connectivity – hence, we use the corresponding property counting the inner l-paths; 2) incoming cross-edges – hence, we cumulate the l-labeled in-degrees of HN vertices; or 3) outgoing cross-edges – hence, we cumulate the number of outgoing l-paths. To handle the ϵ-label in l^*, we additionally estimate, as before, the number of nodes in $\hat{\mathcal{G}}$.
Disjunction. We treat each possible configuration, on both labels. Hence, depending on each case, we cumulate the number of HN inner-edges, on either label, or the cross-edge weights with either label.
Binary Conjunction. We consider all cases, depending on whether: 1) the concatenation label $l_1 \cdot l_2$ appears on a path *within* a HN, 2) one of the labels l_1, l_2 occurs on a HN inner-edge and the other, as a cross-edge, or 3) both labels occur on cross-edges.

EXAMPLE 6. *We evaluate the AQP-translation of example queries of each of the types mentioned above:*

$$[\![l_5]\!]_{\hat{\mathcal{G}}} = Q_L^T(l_5) = \sum_{\hat{v} \in \hat{\mathcal{V}}} EWeight(\hat{v}, l_5) * LPercent(\hat{v}, l_5)$$
$$= EWeight(HN_2, l_5) * LPercent(HN_2, l_5) = 7$$
$$[\![l_2?]\!]_{\hat{\mathcal{G}}} = Q_L^T(l_2) + \sum_{\hat{v} \in \hat{\mathcal{V}}} \mathcal{V}^*Weight(\hat{v}) * \mathcal{V}Weight(\hat{v}) = 27$$
$$[\![l_0^+]\!]_{\hat{\mathcal{G}}} = \sum_{\hat{v} \in \hat{V}} LReach(\hat{v}, l_0) + \sum_{\hat{e} \in \hat{E}} EWeight(\hat{e}, l_0) = 15$$
$$[\![l_0^*]\!]_{\hat{\mathcal{G}}} = [\![l_0^+]\!]_{\hat{\mathcal{G}}} + \sum_{\hat{v} \in \hat{\mathcal{V}}} \mathcal{V}^*Weight(\hat{v}) * \mathcal{V}Weight(\hat{v}) = 40$$
$$[\![l_4 + l_1]\!]_{\hat{\mathcal{G}}} = [\![l_4]\!]_{\hat{\mathcal{G}}} + [\![l_1]\!]_{\hat{\mathcal{G}}} = 14 \text{ and } [\![l_4^- \cdot l_1]\!]_{\hat{\mathcal{G}}} = 7.$$



(a) Grouping Phase

(b) Evaluation Phase

(c) Source and Target Merge Phases

Figure 5: Summarization Phases for $\mathcal{G}_{SN}$

4.2 Query Learning

The problem of learning a regular path query $q \in$ RPQ consists of deriving a query statement from a set of user examples, specified under the form of positive and negative labels on the nodes of an input graph instance. A positive example is thus a positively labeled node n_+ in the input graph $\mathcal{G}$ if n should be present in the query result, while a negative example n_- is the opposite. We denote by S_+ the set of positive examples and by S_- the set of negative ones and their union by $S = S_+ \cup S_-$. In this work, we exemplify the query to be learned as an automaton. Regular languages can alternatively be represented by automata. We refer to [28] for standard definitions of *nondeterministic finite word automaton* (*NFA*) and *deterministic finite word automaton* (*DFA*). We rely on a Gold-style learning algorithm [25], thus on the standard framework of *language identification in the limit*. We aim at a polynomial query learning algorithm that is at the same time sound and complete. Soundness means that given the set of positive and negative examples, the algorithm will correctly return a consistent query with respect to the input positive and negative examples. The algorithm should also be complete, in the sense that it should be capable of learning any query from the set of input examples. It is easy to see that soundness is difficult to achieve, due to the intractability of consistency checking [13, 12], which is PSPACE-complete for queries in RPQ and NP-complete for concatenations of symbols.

Example 7. *Consistency checking for a query of the kind* $Q(l_4, l_5) \equiv l_4 \cdot l_5(\_,\_)$ *corresponding to the* direct reach *of a company via its page ads (i.e., the node pairs of customers and product advertisements in Figure 1) is already NP-complete.*

Due to this intractability, in our work [13, 12] we lifted the soundness condition of the algorithm and resorted to query learning with an abstain condition. If a consistent query cannot be efficiently found, the algorithm abstains from answering. The learning model with abstain is guaranteed to return, in polynomial time, either a consistent query or a null value, if such a query cannot be found. In particular, if a polynomial characteristic sample is provided, the learning algorithm is guaranteed to return the goal query.

The learning algorithm works by selecting the *smallest consistent paths* (SCPs) of length bounded by k (in order to avoid the enumeration of infinite paths). It then generalizes the SCP by states merge on the automaton [35].

The learnability of our query class corresponding to $\text{RPQ}^{\leq n}$ (denoting the RPQ of size at most n) is stated by the following result.

Theorem 3. *The query class* $\text{RPQ}^{\leq n}$ *is learnable with abstain in polynomial time and data, using the algorithm learner with the parameter* k *set to* $2 \times n + 1$.

In order to illustrate the underpinnings of our learning algorithm, we define the notion of a *consistent path* as follows. A path is consistent if it can be selected by the algorithm, for each positive node, and it does not cover any negative node. One can enumerate consistent paths (according to the canonical order $\leq$) by identifying the paths of each node labeled as positive and stopping when a consistent path is found, for each node. We refer to the obtained set of paths as the set of *smallest consistent paths* (SCPs).

EXAMPLE 8. *For example, given the graph in Figure 1 and a sample s.t.* $S_+ = \{C_2, C_{10}\}$ *and* $S_- = \{M_5\}$*, we obtain the SCPs* $l_0 \cdot l_6$*, for* C_{10}*, and, respectively,* $l_0 \cdot l_0 \cdot l_0 \cdot l_6$*, for* C_2*.*

Notice that in this case the disjunction of the SCPs (i.e., the query $l_0 \cdot l_6 + l_0 \cdot l_0 \cdot l_0 \cdot l_6$) is consistent with the input sample and one may think that a learning algorithm should return such a query. The shortcoming of such an approach is that the learned query would be always very simple, in the sense that it uses only concatenation and disjunction. Since we want a learning algorithm that covers all the expressibility of RPQ (in particular including the Kleene star), we need to extend the algorithm with a further step, namely the generalization. The PTA (Prefix Tree Acceptor) [21] of the previous SCPs can be constructed and its states are tentatively merged, if the obtained DFA does not select any negative node. In our example, it is easy to see that the generalized path expression should correspond to $l_0^+ \cdot l_6$.

Although Theorem 3 provides a theoretical k in order to guarantee learnability of queries of a certain size, our practical evaluation [13, 12] showed that small values of k (ranging between 2 and 4) are enough to cover many notable cases of graph query learning.

The above setting is static since the set of positive and negative examples is provided beforehand and no interaction with the user takes place during the learning process. An alternative, *interactive scenario*, can be envisioned that leads to a learning algorithm that starts with an empty sample and continuously interacts with the user during the construction of the input sample. The user provides positive and negative labels on the nodes of the input graph $\mathcal{G}$ until she is satisfied with the output of the learned query. Thus, the sample keeps growing until at most one query consistent with the user's labels is found. This scenario is inspired by the Angluin's model *learning with membership queries* [4].

Let S be a sample over a graph $\mathcal{G}$, the set of all queries consistent with S over $\mathcal{G}$ is defined as:

$$\mathcal{C}(\mathcal{G}, S) = \{q \in \text{RPQ} \mid S_+ \subseteq q(\mathcal{G}) \wedge S_- \cap q(\mathcal{G}) = \emptyset\}.$$

Assuming that the user labels the nodes consistently with some goal query q, the set $\mathcal{C}(\mathcal{G}, S)$ always contains q, where, initially, $S = \emptyset$.

Therefore, an ideal strategy of presenting nodes to the user is able to get us quickly from $S = \emptyset$ to a sample S s.t. $\mathcal{C}(\mathcal{G}, S) = \{q\}$. In particular, a good strategy should not propose to the user the *certain nodes* i.e., nodes not yielding new information when labeled by the user. Formally, given a graph $\mathcal{G}$, a sample S, and an unlabeled node $\nu \in \mathcal{G}$, we say that ν is *certain* (w.r.t. S) if it belongs to one of the following sets:

$$Cert_+(\mathcal{G}, S) = \{\nu \in \mathcal{G} \mid \forall q \in \mathcal{C}(\mathcal{G}, S).\ \nu \in q(\mathcal{G})\},$$
$$Cert_-(\mathcal{G}, S) = \{\nu \in \mathcal{G} \mid \forall q \in \mathcal{C}(\mathcal{G}, S).\ \nu \notin q(\mathcal{G})\}.$$

In other words, a node is certain with a label α if labeling it explicitly with α does not eliminate any query from $\mathcal{C}(\mathcal{G}, S)$.

The notion of certain nodes is inspired by possible world semantics and certain answers [29], and already employed for XML querying for non-expert users [20] and for the inference of relational joins [14, 15]. Additionally, given a graph $\mathcal{G}$, a sample S, and a node ν, we say that ν is *informative* (w.r.t. S), if it is neither labeled by the user nor certain.

An intelligent strategy should propose to the user only informative nodes. Since deciding the informativeness of a node is intractable, we need to explore *practical strategies* that efficiently compute the next node to label. The basic idea behind these is to avoid the intractability of deciding the informativeness of a node, by only looking at a small number of paths of that node. More precisely, we say that a node is *k-informative*, if it has at least one path of length at most k that is not covered by a negative example. If a node is k-informative, then it is also informative, otherwise we are not able to establish its informativeness w.r.t. the current k.

5. CONCLUSION AND PERSPECTIVES

In addition to our overview of topics we addressed concerning graph query evaluation, approximate processing and learning, we also would like to briefly outline the challenges encountered when tackling the problems of query benchmarking and log analysis, as reported in our previous work [8, 18]. In gMark [8], we addressed the graph and query workload generation problems concerning edge-labeled graph instances and UC2RPQ query workloads. Such a generator has been employed in the experimental evaluation of [16, 23] to generate varied test cases of these engines. Benchmarking is a long-standing question in our community, which serves, at the same time, the purpose of both system-driven and theoretical research. On the other hand, empirical analysis of real-world SPARQL query logs [18] also brought to our attention specific query language fragments adopted in practice by users and bots. A major future challenge is to let these studies influence the design of graph query benchmarks that take into account the requirements of users and

applications, as reflected by concrete usage of graph query languages.

As discussed in Section 2, the efforts to understand and hone in graph query expressivity have benefited from the purely-declarative, logic based formulation provided by Datalog. Various of its fragments have been tailored to user-specific applications, with Regular Datalog having recently emerged as an optimal compromise between usability and tractability. In this setting, we provide, as presented in Section 3, a mechanically certified specification of this query language, as well as a custom algorithm for its evaluation and fine-grained incremental maintenance in the dynamic setting. Additionally, we build on state-of-the-art theorem proving technology to verify, with the Coq theorem prover, the correct behavior of the engine. As shown in [16], its reasonable performance on realistic, synthetically-generated [8] graph instances ascertains the potential of employing formal methods to obtain correct-by-construction query engines.

Various works, such as [32, 9, 33], have tackled the complexity of evaluating graph queries, highlighting the challenges it poses. To mitigate these, in Section 4, we explore alternative processing approaches. A first such technique is based on approximate query evaluation, as introduced in [23]. Inspired by the internalization of transitive closure, which lies at the core of Regular Queries, we define a graph summarization that seeks to compress the nodes in the same label-connectivity closure. We combine the compactness of the obtained representation with the expressivity of the property graph model, used to store reachability preservation information, to maximize both evaluation efficiency and accuracy. With respect to existing related approaches, we base our work on the property graph model, leveraging aggregate pre-computation and query-driven graph summarization to provide scalable, high-accuracy, in-database query answers.

In Section 4.2, we have described a polynomial algorithm for learning graph queries of the basic RPQ class. More expressive fragments, for example those including conjunctions, can benefit from our previous work on learning relational queries [14]. A possible unification between the two lines of research would be desirable given the actual occurrences of C2RPQ in real-world query logs [18]. Another direction of future work would be to actually ameliorate the interactive paradigm presented in [14, 13, 12]. In these instances, the initial sample is empty and the user gradually fills the sample by providing positive and negative labels, until the inferred query and the user goal query coincide. For instance, more feedback from the learning system would be needed to more accurately model the user's intentions and to more efficiently reduce the search space given by the initial sample, through asking questions [5].

6. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] R. Angles. The property graph database model. In *AMW*, volume 2100 of *CEUR Workshop Proceedings*, 2018.

[3] R. Angles, M. Arenas, P. Barceló, A. Hogan, J. L. Reutter, and D. Vrgoc. Foundations of modern query languages for graph databases. In *ACM Computing Surveys*, volume 50, pages 68:1–68:40, 2017.

[4] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[5] A. Arioua and A. Bonifati. User-guided repairing of inconsistent knowledge bases. In *EDBT*, pages 133–144, 2018.

[6] P. B. Baeza. Querying graph databases. In *PODS*, pages 175–188, 2013.

[7] G. Bagan, A. Bonifati, R. Ciucanu, G. H. L. Fletcher, A. Lemay, and N. Advokaat. Generating Flexible Workloads for Graph Databases. *PVLDB*, 9(13):1457–1460, 2016.

[8] G. Bagan, A. Bonifati, R. Ciucanu, G. H. L. Fletcher, A. Lemay, and N. Advokaat. gMark: Schema-driven generation of graphs and queries. *IEEE Transactions on Knowledge and Data Engineering*, 29(4):856–869, 2017.

[9] G. Bagan, A. Bonifati, and B. Groz. A Trichotomy for Regular Simple Path Queries on Graphs. In *PODS*, pages 261–272. ACM, 2013.

[10] F. Bancilhon, D. Maier, Y. Sagiv, and J. D. Ullman. Magic Sets and Other Strange Ways to Implement Logic Programs. In *PODS*, pages 1–15, 1986.

[11] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia - A crystallization point for the Web of Data. *Journal of Web Semantics*, 7(3):154–165, 2009.

[12] A. Bonifati, R. Ciucanu, and A. Lemay. Interactive Path Query Specification on Graph Databases. In *EDBT*, pages 505–508, 2015.

[13] A. Bonifati, R. Ciucanu, and A. Lemay. Learning Path Queries on Graph Databases. In *EDBT*, pages 109–120, 2015.

[14] A. Bonifati, R. Ciucanu, A. Lemay, and S. Staworko. A Paradigm for Learning Queries on Big Data. In *Data4U@VLDB*, page 7, 2014.
[15] A. Bonifati, R. Ciucanu, and S. Staworko. Learning Join Queries from User Examples. *ACM Transactions on Database Systems*, 40(4):24:1–24:38, 2016.
[16] A. Bonifati, S. Dumbrava, and E. J. G. Arias. Certified Graph View Maintenance with Regular Datalog. *Theory and Practice of Logic Programming*, 18(3-4):372–389, 2018.
[17] A. Bonifati, G. H. L. Fletcher, H. Voigts, and N. Yakovets. *Querying Graphs*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2018.
[18] A. Bonifati, W. Martens, and T. Timm. An Analytical Study of Large SPARQL Query Logs. *PVLDB*, 11(2):149–161, 2017.
[19] A. K. Chandra and P. M. Merlin. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *STOC*, pages 77–90, 1977.
[20] S. Cohen and Y. Weiss. Certain and possible XPath answers. In *ICDT*, pages 237–248, 2013.
[21] C. de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, 2010.
[22] R. Delanaux, A. Bonifati, M. Rousset, and R. Thion. Query-based linked data anonymization. In *ISWC*, pages 530–546, 2018.
[23] S. Dumbrava, A. Bonifati, A. Ruiz-Diaz, and R. Vuillemot. Approximate Query Processing for Label-Constrained Reachability Queries. *CoRR*, abs/1811.11561, 2018.
[24] O. Erling, A. Averbuch, J. Larriba-Pey, H. Chafi, A. Gubichev, A. Prat-Pérez, M. Pham, and P. A. Boncz. The LDBC social network benchmark: Interactive workload. In *SIGMOD*, pages 619–630, 2015.
[25] E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.
[26] G. Gonthier, A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot, S. L. Roux, A. Mahboubi, R. O'Connor, S. O. Biha, I. Pasca, L. Rideau, A. Solovyev, E. Tassi, and L. Théry. A Machine-Checked Proof of the Odd Order Theorem. In *ITP*, pages 163–179, 2013.
[27] A. Gupta, I. S. Mumick, and V. S. Subrahmanian. Maintaining Views Incrementally. *SIGMOD Record*, 22(2):157–166, 1993.
[28] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.
[29] T. Imielinski and W. Lipski Jr. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, 1984.
[30] A. P. Iyer, A. Panda, S. Venkataraman, M. Chowdhury, A. Akella, S. Shenker, and I. Stoica. Bridging the GAP: towards approximate graph analytics. In *GRADES*, pages 10:1–10:5, 2018.
[31] H. V. Jagadish, R. Agrawal, and L. Ness. A Study of Transitive Closure As a Recursion Mechanism. In *SIGMOD*, pages 331–344, 1987.
[32] K. Losemann and W. Martens. The complexity of regular expressions and property paths in SPARQL. *ACM Transactions on Database Systems*, 38(4):24:1–24:39, 2013.
[33] W. Martens and T. Trautner. Evaluation and Enumeration Problems for Regular Path Queries. In *ICDT*, pages 1–21, 2018.
[34] Michel Dumontier and Alison Callahan and Jose Cruz-Toledo and Peter Ansell and Vincent Emonet and François Belleau and Arnaud Droit. Bio2RDF Release 3: A larger, more connected network of Linked Data for the Life Sciences. In *ISWC*, pages 401–404, 2014.
[35] J. Oncina and P. García. Inferring regular languages in polynomial update time. *Pattern Recognition and Image Analysis*, pages 49–61, 1992.
[36] J. L. Reutter, M. Romero, and M. Y. Vardi. Regular queries on graph databases. *Theory of Computing Systems*, 61(1):31–83, 2017.
[37] The Coq Development Team. The Coq proof assistant, version 8.8.0. `https://zenodo.org/record/1219885`, 2018.
[38] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.
[39] M. Y. Vardi. A theory of regular queries. In *PODS*, pages 1–9, 2016.
[40] Vrandečić, Denny and Krötzsch, Markus. Wikidata: A Free Collaborative Knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

Andrew Chien Speaks Out on the Impact of Current Trends in Architecture

Marianne Winslett and Vanessa Braganholo

Andrew Chien
http://people.cs.uchicago.edu/~aachien/

Welcome to ACM SIGMOD Records series of interviews with distinguished members of the database community. I'm Marianne Winslett, and today, we're at the 2017 SIGMOD and PODS Conference in Chicago. I have here with me Andrew Chien who's a professor at the University of Chicago. Andrew is an ACM fellow, an IEEE fellow, and a fellow of the American Association for the Advancement of Science. Before joining Chicago, Andrew spent five years as the vice president/director of Intel Research. Andrew's Ph.D. is from MIT.

So, Andrew, welcome!

Nice to be here.

You're from outside the SIGMOD community, a highly successful systems researcher. From your outside perspective, what words of wisdom do you have for us data and information researchers?

That's a very interesting question, Marianne. It's a high bar for an outsider, actually.

Well, let me narrow it down a little bit. Can you tell us what, from your perspective, either we've done wrong so far or important problems we haven't given enough attention to yet?

So, yeah, I think the SIGMOD community, actually, is an amazing, dynamic community, and as I look across computer science, several of the great strengths of the community is its deep attachment both to applications as well as to underlying technology and how that changes the game periodically. I also think – I mean, it's pretty obvious to everyone that the growing importance of data and computing systems writ large puts the SIGMOD community sort of in the driver's seat for all kinds of secular change in computing systems.

So, the strengths of the community, I think, are evident, and if I had any advice to give, what I would say is that it seems to me that the SIGMOD community is unique in Computer Science in that it has thought of computation and data in an integrated form. That is, you talk about queries and computation and transactions with a data model, with a schema, with some notion of the structure of the data. Then, you talk about consistency of those kinds of data collections and so on, again in that integrated view of consistency with respect to transactions or computations or workloads or even applications. I actually hope that and would wish that the SIGMOD community would actually try and take some of its learnings and not straightforwardly transliterate them into other fields, into other areas of computer science, but I think that the community has a lot to contribute to the broader space of computing systems.

I spent many of my years as a computer architect, and I can tell you that the issues of data locality, data orchestration, efficient location of computation, efficient parallelization are all fundamental problems in computing systems of every type today, and I can't think of any community that understands better how to combine data organization with consistency with computation than the SIGMOD community. So, I think there are vast contributions to be made by the community.

I think I understand what you're getting at, but can you give me an example of something you could imagine us deciding to do?

I don't know if it's a collective thing. It's sort of can some members of the SIGMOD community that have this knowledge decide to forge out and create new kinds of computational systems that go a bit further from just the data part up into – you're starting to see this, I guess, in certain kinds of – learning systems. Perhaps some examples of systems that are data-centric but actually are much more encompassing than just the data management tasks, so reaching out to include much more complex forms of computation, much more compute-intensive kinds of data transformation and analysis as an integral part of what you might think of as the data management system.

Does Spark Streaming count, or Storm?

I think those systems are examples of movement in that direction. I don't know that they go very far. What do I mean by that? I think that, as I understand those systems – we're fortunate to have an expert in those systems at Chicago now (we have Mike Franklin there now) – there's more of a divide, actually, between the parts of the system around which you have strong properties of consistency or the like and the part that's doing the computation.

I think those systems, as I understand them, are behaving mostly like integration platforms, rather than trying to extend the semantics and the properties and the capabilities and analysis of the underlying database system or data management system up into those computational domains.

So, you'd like to see stronger, for example, consistency guarantees or fault recovery properties in other types of systems as well. Are you thinking more up to new applications or down, putting them at lower levels of the system?

I think it's both directions, actually. I think that you'd like to have those nice properties evident at higher application layers, as you suggested. I think it's also true one of the great successes of the database community is concurrency management as embodied with this notion of transactions, data orchestration, very efficient data movement, and aggregation of different operators, standard query across the memory hierarchy and understanding how to organize that data and that movement and how to represent that data.

Those kinds of things, I think, are in their infancy in computer architecture and in systems, and when you look at the proliferation of nonvolatile storage all over these systems, new kinds of memories – and I'm happy to talk more about that – the opportunities to do

intelligent things are vast, but the frameworks and the understanding for how to do things that are not just locally intelligent but globally intelligent, I think, are lacking.

> ***[…] the issues of data locality, data orchestration, efficient location of computation, efficient parallelization are all fundamental problems in computing systems of every type today, and I can't think of any community that understands better how to combine data organization with consistency with computation than the SIGMOD community.***

What future trends in the lower levels of the system stack in the hardware world do we need to know about that we don't already know about?

Gosh, let me try to address that by rattling off some of the things that I think that are exciting that are happening in the hardware world right now. I think that, starting from several years ago, it's become increasingly clear that specialization is the order of the day. So, what does specialization mean? Well, I don't mean GPUs. GPUs are now a 10-year-old specialization trend. But now, I think you're starting to see pushed both from the bottom – that is in mobile devices – but also from the top – large-scale cloud systems – customization of architectures for higher performance, for energy efficiency, for density, lower latency.

So, that's beginning in the forms of things like FPGAs (Field Programmable Gate Array), Microsoft's Catapult project, and Google's TPU (Tensor Processing Units). TensorFlow processing recently got a lot of press. But it's also true that anyone working in a large-scale vertical application now has the means and the capability and the economics, actually, to do architectural specialization.

So, what does that mean? I think you can expect to see accelerators for almost any focused, large-scale transformation you might want to do. There are the simple things like compression and crypto, which everyone is aware of, but you might imagine indexing, certain kinds of parsing tasks, certain kinds of data representation tasks. Transformations will be accelerated by new kinds of architectural features in the future. That has implications for software. That has implications for query optimization, perhaps, and the costs of different operations, and perhaps it has implications in the long-run for how people formulate their applications.

Do you mean they're accelerated by being put on FPGAs and TPUs, or do you mean some other way of accelerating?

Oh, I view FPGAs as a halfway house. It's a way of balancing the cost of putting custom silicon into these systems, yet preserving some of the breadth and flexibility by allowing it to be reprogrammed. I think what you're going to see over the next couple of years is more and more custom silicon, hardwired silicon, being put onto these chips as SOCs, as instruction set extensions. Oracle has already done some interesting things in their Sonoma series of processors, Sparc-based, that didn't get, I think, a huge amount of press because that's a narrow kind of exposure these days.

But I think those things are happening because they give benefits of 10x, sometimes 100x efficiency in those tasks, and that's just getting too big to ignore, and we have so much silicon. The latest chips being released have 20 billion transistors on computing chips, so there's a lot of room to put interesting stuff on there.

So, for researchers, unless you're in a company who happens to produce these specialty chips, how would one do research on that topic?

It's a good question. I think FPGAs are a vehicle for doing research on these kinds of things, and there are systems available like the NSF Chameleon system that has a set of FPGAs deployed where you could do experiments. Amazon and the other cloud guys.

Whoa! The FPGA is no problem, but the actual custom hardware is going to be way faster than the FPGA. That was what I was getting at.

I think that's a difficult challenge. So, when I was at Intel, the timescale from conception for new architecture features to them appearing in silicon in products was four to five years, but increasingly, in this new world, we're seeing that distance being more like 18 months, 24 months, something like that. So, I think it's possible to work with folks with FPGAs and with simulation, and then, shortly thereafter, with actual hardware deployed at scale in these cloud centers or the like.

So, I don't have any magic bullet that makes it possible to have this time machine and deal with these future hardware systems today before they exist, but if there's any encouragement, it's that the timescales are much shorter. It isn't the time from when you're a graduate student to when you're a full professor when the ideas actually get out there. It's more like two or three SIGMOD review cycles.

Okay, sounds great.

There's a couple of other things on that front. I think there's a number of other trends that are happening that also are disruptive and play to this idea that folks who've been thinking about computation coupled with data coupled with efficient data movement have a lot to contribute.

First, the memory hierarchies in these systems are getting much deeper, and that's despite the fact that processor clock rates aren't getting any faster. What's happening is that multi-core requires more and more bandwidth, and in order to meet these bandwidth needs, people are moving to exotic stacked DRAM kinds of technologies.

Perhaps you've heard of HBM or HBM2 or HBM3. These are stacked DRAM technologies that allow you to get into the terabytes range of memory bandwidth. These represent a super-fast but small memory hierarchy. So, you might have 16 gigabytes, 30 gigabytes, those kinds of numbers, and then, beyond that, you can have the super-large DDR kinds of DRAMs, terabytes or those kinds of things, but you're not going to have terabytes of this super-small, fast memory.

So, if you're thinking memory hierarchies go away, they probably don't go away. What you're seeing is one grows, and that means that that gives birth to another smaller one that's above it. These memory hierarchies not only have bandwidth differences, so you might have terabytes per second of bandwidth in this small stacked memory hierarchy. That might actually mean that at the DDR level or in this future persistent 3D XPoint or other kinds of nonvolatile memory, you might have even less memory bandwidth there. You might have 100 gigabytes per second or less because the introduction of a new tier usually for architects is an excuse to reduce bandwidth and performance at lower layers.

So, I think that's a challenging problem. Beyond that, of course, we have the widespread acceptance of SSDs and flash-based things. So, you've got at least three or four interesting tiers of memory hierarchy that seem to be here for the foreseeable future.

Beyond that, there are all kinds of opportunities to do interesting things between them because the gap isn't so large. I think the storage management community, the data community, has been, for many years, to a degree, shaped by the large gap between disk and DRAM.

Not anymore.

And now we have flash and NAND.

We went main memory a while back because key customers tend to have problems that fit in main memory, so you've got to rethink everything from scratch. So, we went to the app route. And caching and buffering is always a popular topic, but I don't know that we've gone for quite as many layers as you've been talking about yet. I don't know.

Okay, well, that's good to hear. So, that's one dimension of what's happening. Another thing that has happened is that in this push to create the DRAM replacement, the hardware technology community has produced a whole bunch of different kinds of memories, and some of them, actually, are quite a bit faster and more reliable and more persistent than some of the technologies being pushed as DRAM replacements.

So, just to hold out a few examples, there's MRAM technologies and other kinds of exotic memory types that, while not cheap enough that you would ever dream of replacing all of your SRAM or all of your DRAM with those technologies, they can be used in spots and different special functions.

So, I know there are some researchers looking at this, but I think there's probably a larger opportunity to exploit those kinds of special memory technologies for narrower uses, perhaps certain kinds of locking structures, certain kinds of logging structures, things related to performance-critical aspects.

Do you think that, in the nodes in the cloud, there'll be little bits of these specialty memories sprinkled around?

I think that's likely, and I think it's also likely you may see little bits of the specialty memory actually integrated into compute chips in the future, so that would be another way it could become generally accessible.

I was going to add one more thing about memories. There's a lot of exciting stuff happening in memory systems, perhaps more change now than there has been in decades. Another thing that's happening is this vision around disaggregated servers or this looser association between memories and CPUs. For a long time, we've had this traditional notion of either pizza boxes – processor, DRAM, and maybe some disk or maybe some other kind of storage and scale out in the cloud – and now, increasingly, you have fat nodes and other kinds of pairings of quantities of DRAM and network-attached storage and other kinds of things.

Sort of the natural and logical extension of that that's being enabled by super-high-speed networks and interconnects is disaggregated resources in the cloud. So, you can imagine large nodes that are close to memory only. You can imagine nodes that are primarily compute and coupled to large shared pools of memory that might be shared amongst multiple different domains of processing.

So, this raises a bunch of really interesting questions about how do you manage irregular memory performance, distributed memories of different capacities, various associations of computing, and all the data locality problems are different now in this space. And I think that we're perhaps well-prepared from the distributed systems and cloud software side to think about those problems, but understanding how to do it well, I think, requires this integrated data model and computation view that has been one of the cores of the database community for years.

Super. What major trends at the application level might have escaped our attention?

Gosh, I'm loathe to presume that they've escaped this database community's attention, but some of the things I think that are big changes, compared to what I hear about the database community focused on, is there's this explosive growth around how data relates to society, that is, geographic, national boundaries, regulatory boundaries, different commercial proprietary boundaries, and so on, and it seems like that's increasingly a fundamental aspect of function and performance of these systems. And while I don't pretend to know everything that's going on in the database community, it seems to me that those aren't traditional foci of the community and I think are important challenges for how this all goes forward.

The second area that I would comment on is there's excitement around IoT – Internet of Things – self-driving cars, any other kinds of network or edge devices that might be high data-rate sources. So, there are a couple different ways to think about those systems. I hear a lot about people thinking about the cloud side or the server side element of those problems, "After I've uploaded a lot of that data, how do I do analytics on it? How to do real-time analytics on it? How do I store and organize it and so on?" But the reality is that for lots of those systems, large fractions of the data will never make it to the cloud. One of the funny secrets of the sensor systems is that most video cameras don't bother capturing, actually, most of their data. Certainly, the self-driving cars, where they're talking about data rates of many gigabytes per second while the vehicles are in operation, imagine multiplying that by tens of millions of these vehicles. It's not hard to figure out that you can't actually afford to capture, network, and store all of that stuff for very long, if it ever gets to the data center.

So, there's a bunch of interesting challenges about how do you do distributed and streaming data analytics, how do you deal with collections that are fundamentally asymmetrically distributed, and how do you host applications in some reasonable programming and performance-tuning model across those very, very complicated kinds of infrastructures. And then, you've got all the security and privacy and governance kinds of questions we talked about before.

You should realize that the process of research is to make a contribution to the intellectual direction of the community. If you're completely aligned with the intellectual direction of the community, you're not making any contribution to the vector. Maybe a small contribution to magnitude, but not direction.

We had some very interesting conversations with a provider here in Chicago that turns out to be the host of a lot of self-driving or connected-car kinds of applications. It's the company that was formerly called Navteq, bought by Nokia and then sold to an alliance of German auto manufacturers. Very interesting company. And the challenge they have is how do they service these different companies that are fundamentally competitors, and of course they all have their own data. They all would benefit from some kind of data pooling because some of the companies are exotic, high-end kinds of firms that don't sell tens of millions of cars a year and don't have the same coverage. Others are that, but don't necessarily have the high-end sensing platforms in their vehicles for economic reasons.

So, those kinds of companies and venues have all of these problems today, and I can tell you from talking to them, they don't have good systems solutions to address the privacy needs, the sharing needs, and the vertical aspects of those systems.

Do you have any words of advice for fledgling or mid-career researchers?

That's an interesting challenge. I guess what I would say is what I've learned over the years is I'd encourage them not to pursue any fads. There are lots of fads.

So, how do you know you're not pursuing a fad? I think you need to make sure that the trends that your research ideas or research direction depend on are fundamental, and you have to figure that out for yourself because the fads won't tell you that. And if you look hard and try to understand what the fundamentals are that are driving the importance of a set of research ideas or research direction, when you formulate your research problems and contributions, you're likely to get resistance from the community.

This is not necessarily a bad thing. You should realize that the process of research is to make a contribution to the intellectual direction of the community. If you're completely aligned with the intellectual direction of the community, you're not making any contribution to the vector. Maybe a small contribution to magnitude, but not direction. So, it's not a bad thing if you initially get resistance, and you have to make the case, and you have to make a justification of the problems and so on that you work on. But I think that if you're persistent at it, then that's the way, actually, to make a long-term impact.

And my experience has been we've worked in areas, and I have to confess that for me personally, at times, we had strong convictions about where things were gonna go, and we gave up too soon. And I think that in hindsight, when the community finally came around to those thoughts and ideas, we could have made a larger contribution if we had stuck with it and really built up that critical mass of both understanding as well as evidence and prominence in that area to drive the community forward.

If you magically had enough extra time to do one additional thing at work that you're not doing now, what would it be?

There are a hundred things, but the thing I miss the most, and actually the reason that I decided to come back to the university, was the opportunity to have more time to be hands-on with technology. So, for me today, that means experimenting with new systems that have been put out. That means writing a little code. I never get to write as much as I would like. Maybe it means doing a little hardware design for me. But designing is as much about exploring all of the exciting new things that the community and the industry is producing and understanding what's possible as it is about designing or coding per se.

If you could change one thing about yourself as a computer science researcher, what would it be?

I think I would say, and it's related to the comment I made earlier about advice for young researchers, it would be patience. One of the things I've learned over the years is that you can often figure out a problem – first, you make sure it's a real problem – and produce academic-quality kinds of ideas and solutions and proof. And then, if you build on that, you have a larger mass of proof. But it takes a long, long time for those ideas or systems to actually find their way into large-scale use. The idea is to find their way into the center of the community and so on.

So, I think that we're all, in the research community, very, very smart, quick people, and we need to understand that there's a big gap between understanding at an intellectual level and acceptance and broad dissemination. So, one improvement for me would be to have the patience, actually, to see these things through, to make sure that inventions that we have or good ideas that we have actually have the maximum impact they can have.

Thank you very much for talking with us today.

Thank you for having me.

Data Management Systems Research at TU Berlin

Ziawasch Abedjan, Sebastian Breß, Volker Markl, Tilmann Rabl, Juan Soto
Technische Universität Berlin and DFKI Berlin
{ziawasch.abedjan,sebastian.bress, volker.markl, rabl, juan.soto}@tu-berlin.de

ABSTRACT

Data management systems research at TU Berlin is spearheaded by the Database Systems and Information Management (DIMA) Group, the Big Data Management (BigDaMa) Group, as well as the affiliated Intelligent Analytics for Massive Data (IAM) Research Group at the German Research Center for Artificial Intelligence (DFKI). Jointly, our research activities encompass a wide variety of database topics, including benchmarking, data integration, modern hardware, and scalable data processing.

As of Fall 2018, the current team is comprised of three university professors, thirteen senior and postdoc researchers, twenty PhD students, and several research assistants. Among our notable accomplishments is the DFG-funded Stratosphere Research Unit, which laid the groundwork for what would later become Apache Flink. DIMA has also been leading the Berlin Big Data Center, one of only two BMBF-funded Big Data Competence Centers in Germany since 2014. In addition, DIMA is co-directing the Berlin Center for Machine Learning, one of four BMBF-funded Machine Learning Competence Centers in Germany.

1. INTRODUCTION

Modern applications have to cope with large, fast, and heterogeneous data, bridging the worlds of advanced data analysis and machine learning with data management. Naturally, this poses numerous research challenges for the design and usage of data analytics systems. Fortunately, novel advances in hardware technologies, data flow architectures, and machine learning techniques are making it possible to build efficient and user-friendly data processing systems. With a team of thirty database researchers, comprised of doctoral students, senior researchers, and university professors, TU Berlin is well positioned to address key challenges.

Given the significant importance of data flow systems, at its onset DIMA[1] embarked on the development of a next generation big data analytics platform. Initially, known as Stratosphere [6], over the course of several years, it would later go on to become Apache Flink, an open-source stream processing framework for parallel dataflow analysis.

Today, ongoing research focuses on meeting the requirement needs of novel Internet of Things infrastructures and increasing their ease of use. We have developed a declarative programming interface to enable data scientists to primarily focus their attention on analysis. Other key research topics that are underway include research on modern hardware, systems benchmarking, end-to-end machine learning pipelines, responsible data management, data analysis infrastructures, and information marketplaces. Moreover, the recently established BigDaMa Group[2] is actively conducting research (e.g., building end-to-end data preparation systems) to address data heterogeneity challenges.

In the following sections, we further motivate several of the aforementioned research topics and highlight key contributions from our database systems researchers. We will conclude with an overview of our existing grants and collaboration activities across our research projects.

2. SCALABLE DATA PROCESSING

Many specialized data processing systems have been developed, in order to analyze high *volume*, *velocity*, and *variety* data, efficiently and effectively. To meet these demands, systems often exploit specially optimized libraries. For example, to perform numerical linear algebra operations, conduct natural language processing, or execute graph analytics. Moreover, system building commonly employs both modern storage, such as non-uniform memory access (NUMA) designs and processing architectures, such as heterogeneous CPUs, to achieve higher performance. It is the wide diversity of systems and hardware solutions that greatly improve functionality and reduce the execution time for many data

[1] https://www.dima.tu-berlin.de/

[2] https://www.bigdama.tu-berlin.de

analytics-dependent applications.

Next, we discuss our research contributions in the area of scalable data processing, which includes Stratosphere, Apache Flink, stream processing, and declarative programming.

2.1 Stratosphere and Apache Flink

The official Apache Flink project website [1, 18] declares that *"Apache Flink is an open source platform for distributed stream and batch data processing.* [At its] *core* [it] *is a streaming data flow engine that provides data distribution, communication, and fault tolerance for distributed computations over data streams."*

Many of the original concepts in Stratosphere inspired or were carried over to Apache Flink [8, 23], such as the query optimizer, the streaming dataflow runtime, and the support for iterations. While others remained experimental features, such as the optimistic fault tolerance [32, 48]).

Apache Flink offers numerous additional features. For example, it: (i) provides consistent, exactly once guarantees for event time processing, (ii) is fault-tolerant, even for stateful operations, and (iii) is highly scalable, able to run on thousands of nodes with high throughput and low latency.

Additionally, Apache Flink offers several APIs. For example, the DataStream API applies transformations, such as filtering and aggregation on data streams. The Table API supports the composition of queries from relational operators, such as selection, filter, and join. Furthermore, Flink provides numerous libraries, such as the: CEP library for complex event-processing, FlinkML library for machine learning, and Gelly library for graph-processing. Using Flink's APIs and libraries, software developers are empowered to build and execute applications that run on Flink. Flink is increasingly gaining traction around the world. According to Alibaba [33], it is Flink's distinguishing technological capabilities that make it the *"most advanced stream processing engine today."*

2.2 Stream Processing

In recent years, our researchers have investigated how to enhance Apache Flink and related systems. For example, we devised a novel technique to address performance challenges faced when conducting aggregate sharing in data stream windows. Subsequently, we developed a prototype in Flink and demonstrated that our technique outperforms the state-of-the-art [19, 28].

Moreover, we developed optimizations to improve both the sharing of windows and computation for highly distributed setups [50], interactivity in streaming visualizations [51], and surveyed state management [49]. Currently, we are conducting research on the management of large-state for analytics that are beyond the capabilities of today's batch and stream processing engines.

2.3 Managing the Data Science Process

Reducing the entry barrier and cost of analyzing large amounts of data at scale requires the simplification of the data analysis process, which is today a grand challenge in data management research [42]. Addressing this demand, will require the development of a novel approach to automate the implementation decisions that data scientists routinely make, such as the decisions about the heterogeneous computing environments to employ. Particularly, since they are founded on a broad spectrum of theories, systems, and hardware solutions. Automated optimization, parallelization, and hardware-adaptation is a holy-grail of data science.

This grand challenge can be met, if we combine existing data processing technologies currently available in the scientific and systems community. The major obstacle to achieving automation is the absence of a principled model for scalable data science systems, akin to relational algebra in database systems. The key is to provide a declarative, algebraic, and optimizable representation for the entire data analysis process. To solve this problem, we need to integrate disparate hardware and software components present in today's data analysis architectures into a unified mosaic of systems, hardware devices, and theories that view analytics as graphs, matrices, or relations.

As a first step towards solving the automation problem, our researchers developed Emma [2, 7], a Scala DSL that enables holistic optimizations of data flow programs for scalable data analysis on Apache Flink and Apache Spark. As a result, developers can disregard the details of a platform-specific API, which reduces both program development and execution time. In 2015, Emma garnered an ACM SIGMOD Research Highlight Award.

Our researchers also introduced Lara, a deeply embedded language in Scala that enables developers to exploit optimizing transformations across linear and relational algebra operators [38] and physical operators, such as Blockjoin [39] to bridge relational and matrix representations and write scalable programs. Additionally, we devised a novel approach called ScootR [40] that significantly improves the performance of R programs executed in data flow systems, by establishing bidirectional ac-

cess between native user-defined functions and Flink's data structures. We also developed novel data handling methods, to better cope with large data sets and more efficiently yield visualizations by exploiting data aggregation approaches [34, 35]. The research conducted in this area received a VLDB best paper award in 2014.

Currently, we are investigating how to optimize the entire iterative data science process, from data source selection over information extraction and integration to data analysis, model building, model application, and visualization. Moreover, we are embarking on novel research in the areas of large-scale data analysis infrastructures, data management for the Internet of Things, data processing in the fog, end-to-end machine learning, information marketplaces, and technological enablers for responsible data management.

3. MODERN HARDWARE

The modern hardware landscape is rapidly evolving. Today, there are massively parallel processors with anywhere from hundreds to thousands of cores in graphic processing units (GPU) and the many integrated core (MIC) architecture, whose series of microarchitectures integrate many physical cores onto a single integrated circuit. Additionally, main memory costs have continued to drop, enabling a database to be stored in main memory. Network technologies, such as Infiniband and remote direct memory access (RDMA) provide low latency communication and low network bandwidth on the same order of magnitude as main memory bandwidth, as discussed by Binnig et al. in [10].

These novel technologies can accelerate data management by orders of magnitude, decrease computing costs by scaling-down cluster resources, and reduce the data to knowledge time. We conduct research in the modern hardware space to discover new ways to exploit these technologies.

3.1 Hardware Tailored Query Compilation

The power wall is arguably the defining limit of modern processor performance. Thus, vendors develop processor cores that are specialized to particular tasks, such as *ARM big.LITTLE*, a heterogeneous computing architecture. Alternatively, they develop processors that adhere to a new processor architecture, which is fundamentally different than classical CPUs, as discussed by Borkar and Chien [14] and Esmaeilzadeh et al. [22]. However, for data management systems it is difficult to exploit these heterogeneous processors. Instead, costly experts are required to re-implement and optimize query processors for new processor architectures.

To overcome this challenge, we launched the Hawk Project[3], in order to automatically exploit heterogeneous processors and increase performance in data management systems. The key problem is how to support many heterogeneous processors efficiently without having to rewrite code, for each new processor release. A problem that commonly arises with data management operators. Naturally, such an effort is both costly and error prone. Our aim is to enable data management systems to rewrite their code until they run optimally on a single processor.

W have developed a hardware-tailored code generator called Hawk [15] based on the CoGaDB system [16]. Hawk utilizes advanced query compilation strategies to produce custom code variants for each processor and query. By automatically exploring code variants, Hawk can tune generated code for each processor avoiding manual tuning and sidestepping the need for expensive experts.

3.2 Data Processing on Modern Processors

Over the past few years, we have been investigating alternative ways to leverage heterogeneous processor capabilities. For example, we explored code variants for selections and aggregations using an approach akin to micro adaptivity [46, 47]. We also implemented vectorized hashing primitives (e.g., gather, scatter, selective load, selective store) in OpenCL, to reduce code complexity and enable portability for both CPUs and MICs [9].

In addition, we devised a new approach to execute the k-means [41] algorithm more efficiently on GPUs and achieve a higher throughput (up to 20x over state-of-the-art approaches). Furthermore, we discovered how we can use the GPU memory hierarchy efficiently and developed compilation-based query processing strategies for massively parallel processors [27]. We also experimentally evaluated design aspects of current stream processing systems on modern hardware and found that the throughput of streaming systems on a single node can be improved by up to two orders of magnitude [53]. Our system Ocelot [31] is an OpenCL-based execution engine for the MonetDB main-memory database, which assesses efficiency in systems that completely rely on a hardware-oblivious code base. Finally, we investigated how to accelerate query optimization using massively parallel processors [29, 30, 37].

4. BENCHMARKING

Today's big data systems are designed to be scalable and meant to be run on a large number of

[3] `https://www.dfg-spp2037.de/ma4662-5`

nodes, in order to distribute workloads and speedup processing. Once these systems come to exist, benchmarking them is of paramount importance, to measure their performance under varying real-world scenarios. Historically, the Transaction Processing Performance Council's (TPC) benchmarks have enabled database researchers to meet their optimization goals. With the sheer-diversity of data processing systems under development at an ever-increasing pace, new benchmarking scenarios as well as novel tooling are required to properly assess system performance.

Our research in benchmarking data management systems includes designing, developing, and conducting performance surveys, devising novel measurement techniques and building software tools that implement them, and contributing to standardization efforts. Our objectives include identifying a system's capabilities and limitations as well as providing insight into functional areas, where additional investigation is required. Next, we present our benchmarking tools for big data, stream processing, and machine learning systems.

4.1 Benchmarking Big Data Systems

Our researchers have been instrumental in the development and standardization of several application-level benchmarks, including TPCx-BB [17], TPCx-IOT [44], and TPC-DS [45]. Typically, benchmark projects require a large number of configuration and data collection steps for the many experiments that need to be conducted. To simplify this tedious process, we have developed the Peel [11] framework. It automates the setup and deployment of big data frameworks, conducts benchmark experiments, gathers all system and performance data, and stores them in versioned repositories.

4.2 Benchmarking Stream Processing Systems

In recent years, there has been a surge in the number of novel stream processing systems (SPS). This poses numerous challenges for benchmarking due to the many subcomponents involved. In particular, since these may be outside of the boundary of the system under test and can easily become the predominant bottleneck. Recently, we demonstrated that all of the earlier benchmarking studies for SPS violated assumptions about the system setup. Consequently, since system's do not have control over incoming data streams, our experiments demonstrated [36] that the measurements reported in these studies both overestimated throughput and underestimated latency.

4.3 Benchmarking Machine Learning Systems

Machine learning has become ubiquitous for many data-driven applications. Since there is a natural trade-off between accuracy and performance in machine learning models, solely benchmarking the performance of machine learning systems is insufficient. Thus, we are currently conducting research to develop comprehensive benchmarks and build benchmarking tools that ensure reproducibility for machine learning systems.

The advent of big data processing systems, such as Hadoop and next generation systems, such as Apache Spark have quickly spurred interest in implementing more complex analytics jobs (beyond simple indexing or sorting) on these scalable systems. Among these complex analytics are Apache Mahout and SparkML. Although these libraries similarly feature weak-scaling capabilities as simple processing jobs, they do not scale in other ways (e.g., in terms of model dimensionality [13]). Furthermore, many evaluations today compare their machine learning systems against weak baselines, such as simple and highly-inefficient Hadoop implementations that are easily outperformed by state of the art single-node machine learning libraries [12].

5. DATA INTEGRATION

Increasingly, organizations want to obtain value from their disparate datasets. To do so, they inject all of their data into *data lakes*, to make the data available for analysis. Although this approach solves the data access problem, another challenge for effective data analysis remains: metadata about datasets is often missing or poorly documented and data scientists rarely possess comprehensive knowledge about the data lake. Today, data discovery, data integration, and data cleaning are factually the most time-consuming and least enjoyable tasks for data scientists [20]. In addition to small contributions to the field of entity resolution [21, 43], our research in the area of data integration tackles two general challenges, i.e., data discovery in data lakes and iterative data preparation.

5.1 Data Lake Management

Despite the presence of data lakes, discovering the data of actual interest is still very challenging [25]. Due to the abundance of data without a central owner, a holistic organization of these datasets via ETL is infeasible. As a compromise, a proposed solution is to generate easy to obtain metadata [5] from datasets in the data lake and infer relationships, such as foreign/primary key relationships and

other types of inter-column similarities.

Jointly, with colleagues from Massachusetts Institute of Technology (MIT), Qatar Computing Research Institute (QCRI), and the University of Waterloo, we built *Data Civilizer* [20, 26], an end-to-end big data management system. Data Civilizer incorporates a data discovery component called *Aurum* [24] that achieves the aforementioned functionalities and efficiently identifies column similarities and overlaps. However, these types of heuristics can lead to the generation of many false positives. In particular, numerical columns, such as ID columns are often quite similar. Currently, we aim to solve this problem, by developing new heuristics that disambiguate those columns more accurately.

5.2 Iterative Data Cleaning

While there has been a huge body of work in the area of data cleaning, most data practitioners resort to custom data wrangling scripts. The main reason behind this is that there is still no one-size-fits-all system for data cleaning [4]. Algorithms tackle very specific types of errors, such as rule or pattern violations and outliers. As a result, the data scientist will undergo an iterative try-and-error procedure, which is time-consuming. Additionally, most of these algorithms require some sort of hyperparameter or a set of given patterns/rules, which may be unavailable. To bridge this gap, we treat data cleaning as an iterative process and preserve the history of previously performed cleaning tasks, to minimize the overall user-effort and identify the right set of cleaning routines and their respective configurations for the task at hand. Furthermore, have studied several aggregation methods to combine the effectiveness of varying error detection strategies [52].

6. GRANTS, ALLIANCES, AND SERVICE

Our research activities are funded through grants obtained from varying national, international, and industry sources, including the German Federal Ministry of Education and Research (BMBF), the German Federal Ministry for Economic Affairs and Energy (BMWi), the German Federal Ministry of Transport and Digital Infrastructure (BMVI), the German Research Foundation (DFG), and the European Union, among others. Most notably, we are coordinating two German flagship big data projects, the Berlin Big Data Center[4] (BBDC) and the Smart Data Forum[5] as well as co-directing the Berlin Center for Machine Learning (BZML).

We have transferred our research into numerous commercial products and open-source systems. We closely collaborate with many leading information management companies and have created several startups based on our research. We also contribute to our governmental and scientific communities. For example, we serve as grant reviewers and on expert panels at national and international funding agencies and provide expert advice to government agencies in Germany and the EU.

Our researchers support the database community on various levels. In 2013, Volker participated in the Beckman Database Research Self-Assessment Meeting to discuss the state of database research and offer perspectives on key directions for future research [3]. Since 2018, Volker is President of the VLDB Endowment. Furthermore, we have hosted an EDBT conference and served as conference officers and program committees for VLDB, SIGMOD, and ICDE, among others.

[4] `https://www.bbdc.berlin`

[5] `https://smartdataforum.de`

7. REFERENCES

[1] Official apache flink website. `https://flink.apache.org/`.

[2] Official website of the emma language. `https://github.com/emmalanguage/emma`.

[3] D. Abadi, R. Agrawal, et al. The beckman report on database research. *Communications of the ACM*, 59(2):92–99, Jan. 2016.

[4] Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang. Detecting data errors: Where are we and what needs to be done? *PVLDB*, 9(12):993–1004, Aug. 2016.

[5] Z. Abedjan, L. Golab, and F. Naumann. Profiling relational data: a survey. *VLDBJ*, 24(4):557–581, 2015.

[6] A. Alexandrov et al. The stratosphere platform for big data analytics. *VLDBJ*, 23(6):939–964, 2014.

[7] A. Alexandrov, A. Kunft, A. Katsifodimos, F. Schüler, L. Thamsen, O. Kao, T. Herb, and V. Markl. Implicit parallelism through deep language embedding. In *SIGMOD*, pages 47–61, 2015.

[8] D. Battré, S. Ewen, F. Hueske, O. Kao, V. Markl, and D. Warneke. Nephele/pacts: a programming model and execution framework for web-scale analytical processing. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, pages 119–130, 2010.

[9] T. Behrens, V. Rosenfeld, J. Traub, S. Breß, and V. Markl. Efficient SIMD vectorization for hashing in opencl. In *EDBT*, pages 489–492, 2018.

[10] C. Binnig, A. Crotty, A. Galakatos, T. Kraska, and E. Zamanian. The end of slow networks: It's time for a redesign. *PVLDB*, 9(7):528–539, 2016.

[11] C. Boden, A. Alexandrov, A. Kunft, T. Rabl, and V. Markl. PEEL: A framework for benchmarking distributed systems and algorithms. In *TPCTC*, pages 9–24, 2017.

[12] C. Boden, T. Rabl, and V. Markl. Distributed machine learning - but at what cost? In *Workshop on ML Systems at NIPS*, 2017.

[13] C. Boden, A. Spina, T. Rabl, and V. Markl. Benchmarking data flow systems for scalable machine learning. In *Proceedings of the 4th ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond, BeyondMR@SIGMOD 2017, Chicago, IL, USA, May 19, 2017*, pages 5:1–5:10, 2017.

[14] S. Borkar and A. Chien. The future of microprocessors. *Communications of the ACM*, 54(5):67–77, 2011.

[15] S. Breß et al. Generating custom code for efficient query execution on heterogeneous processors. *VLDBJ*, 2018.

[16] S. Breß, H. Funke, and J. Teubner. Robust query processing in co-processor-accelerated databases. In *SIGMOD*, pages 1891–1906, 2016.

[17] P. Cao, B. Gowda, S. Lakshmi, C. Narasimhadevara, P. Nguyen, J. Poelman, M. Poess, and T. Rabl. From bigbench to tpcx-bb: Standardization of a big data benchmark. In *Performance Evaluation and Benchmarking. Traditional - Big Data - Interest of Things - 8th TPC Technology Conference, TPCTC 2016, New Delhi, India, September 5-9, 2016, Revised Selected Papers*, pages 24–44, 2016.

[18] P. Carbone et al. Apache flink^{TM}: Stream and batch processing in a single engine. *IEEE Data Eng. Bull.*, 38(4):28–38, 2015.

[19] P. Carbone, J. Traub, A. Katsifodimos, S. Haridi, and V. Markl. Cutty: Aggregate sharing for user-defined windows. In *CIKM*, pages 1201–1210, 2016.

[20] D. Deng, R. C. Fernandez, Z. Abedjan, S. Wang, M. Stonebraker, A. Elmagarmid, I. Ilyas, S. Madden, M. Ouzzani, and N. Tang. The data civilizer system. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, 2017.

[21] D. Deng, W. Tao, Z. Abedjan, A. K. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. Unsupervised string transformation learning for entity consolidation. In *ICDE*, 2019.

[22] Esmaeilzadeh et al. Dark silicon and the end of multicore scaling. In *ISCA*, pages 365–376, 2011.

[23] S. Ewen, K. Tzoumas, M. Kaufmann, and V. Markl. Spinning fast iterative data flows. *PVLDB*, 5(11):1268–1279, 2012.

[24] R. C. Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker. Aurum: A data discovery system. In *ICDE*, 2018.

[25] R. C. Fernandez, Z. Abedjan, S. Madden, and M. Stonebraker. Towards large-scale data discovery: position paper. In *International Workshop on Exploratory Search in Databases and the Web (ExploreDB)*, pages 3–5, 2016.

[26] R. C. Fernandez, D. Deng, E. Mansour, A. A. Qahtan, W. Tao, Z. Abedjan, A. K. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. A demo of the data civilizer system. In *SIGMOD*, pages 1639–1642, 2017.

[27] H. Funke, S. Breß, S. Noll, V. Markl, and J. Teubner. Pipelined query processing in coprocessor environments. In *SIGMOD*, 2018.

[28] P. M. Grulich, R. Saitenmacher, J. Traub, S. Breß, T. Rabl, and V. Markl. Scalable detection of concept drifts on data streams with parallel adaptive windowing. In *EDBT*, pages 477–480, 2018.

[29] M. Heimel, M. Kiefer, and V. Markl. Self-tuning, gpu-accelerated kernel density models for multidimensional selectivity estimation. In *SIGMOD*, pages 1477–1492, 2015.

[30] M. Heimel and V. Markl. A first step towards gpu-assisted query optimization. In *ADMS@VLDB*, pages 33–44, 2012.

[31] M. Heimel, M. Saecker, H. Pirk, S. Manegold, and V. Markl. Hardware-oblivious parallelism for in-memory column-stores. *PVLDB*, 6(9):709–720, 2013.

[32] F. Hueske, M. Peters, M. Sax, A. Rheinländer, R. Bergmann, A. Krettek, and K. Tzoumas. Opening the black boxes in data flow optimization. *PVLDB*, 5(11):1256–1267, 2012.

[33] X. Jiang. Unified engine for data processing and ai. `https://berlin-2018.flink-forward.org/conference-program/#unified-engine-for-data-processing-and-ai`. Alibaba, Flink Forward, Berlin, 2018.

[34] U. Jugel, Z. Jerzak, G. Hackenbroich, and V. Markl. M4: A visualization-oriented time series data aggregation. *PVLDB*, 7(10):797–808, 2014.

[35] U. Jugel, Z. Jerzak, G. Hackenbroich, and V. Markl. Vdda: Automatic visualization-driven data aggregation in relational databases. *VLDBJ*, 25(1):53–77, 2016.

[36] J. Karimov, T. Rabl, A. Katsifodimos, R. Samarev, H. Heiskanen, and V. Markl. Benchmarking distributed stream processing engines. In *ICDE*, 2018.

[37] M. Kiefer, M. Heimel, S. Breß, and V. Markl. Estimating join selectivities using bandwidth-optimized kernel density models. *PVLDB*, 10(13):2085–2096, 2017.

[38] A. Kunft, A. Alexandrov, A. Katsifodimos, and V. Markl. Bridging the gap: towards optimization across linear and relational algebra. In *BeyondMR@SIGMOD 2016*, page 1, 2016.

[39] A. Kunft, A. Katsifodimos, S. Schelter, T. Rabl, and V. Markl. Blockjoin: Efficient matrix partitioning through joins. *PVLDB*, 10(13):2061–2072, 2017.

[40] A. Kunft, L. Stadler, D. Bonetta, C. Basca, J. Meiners, S. Breß, T. Rabl, J. J. Fumero, and V. Markl. Scootr: Scaling R dataframes on dataflow systems. In *SOCC*, pages 288–300, 2018.

[41] C. Lutz, S. Breß, T. Rabl, S. Zeuch, and V. Markl. Efficient and scalable k-means on GPUs. *Datenbank-Spektrum*, 2018.

[42] V. Markl. Breaking the chains: On declarative data analysis and data independence in the big data era. *PVLDB*, 7(13):1730–1733, Aug. 2014.

[43] Öykü Özlem Çakal, M. Mahdavi, and Z. Abedjan. Clrl: Feature engineering forcross-language record linkage. In *EDBT*, 2019.

[44] M. Poess, R. Nambiar, C. Narasimhadevara, K. Kulkarni, T. Rabl, and H.-A. Jacobsen. Analysis of tpcx-iot: The first industry standard benchmark for iot gateway systems. In *ICDE*, 2018.

[45] M. Poess, T. Rabl, and H. Jacobsen. Analysis of TPC-DS: the first standard benchmark for sql-based big data systems. In *SOCC*, pages 573–585, 2017.

[46] V. Rosenfeld, M. Heimel, C. Viebig, and V. Markl. The operator variant selection problem on heterogeneous hardware. In *ADMS@VLDB*. VLDB Endowment, 2015.

[47] B. Răducanu et al. Micro adaptivity in Vectorwise. In *SIGMOD*, pages 1231–1242, 2013.

[48] S. Schelter, S. Ewen, K. Tzoumas, and V. Markl. "all roads lead to rome": optimistic recovery for distributed iterative data processing. In *CIKM*, pages 1919–1928, 2013.

[49] Q.-C. To, J. Soto, and V. Markl. A survey of state management in big data processing systems. *VLDBJ*, Aug 2018.

[50] J. Traub, S. Breß, T. Rabl, A. Katsifodimos, and V. Markl. Optimized on-demand data streaming from sensor nodes. In *SoCC*, pages 586–597, 2017.

[51] J. Traub, N. Steenbergen, P. Grulich, T. Rabl, and V. Markl. I^2: Interactive real-time visualization for streaming data. In *EDBT*, pages 526–529, 2017.

[52] L. Visengeriyeva and Z. Abedjan. Metadata-driven error detection. In *SSDBM*, 2018.

[53] S. Zeuch, B. Del Monte, J. Karimov, C. Lutz, M. Renz, J. Traub, S. Breß, T. Rabl, and V. Markl. Analyzing efficient stream processing on modern hardware. *PVLDB*, 12(5):516–530, 2018.

SIGMOD 2018 Program Committee Chair's Report

Philip A. Bernstein
Microsoft Research
philbe@microsoft.com

ABSTRACT

This paper reports on the program committee process for SIGMOD 2018, including statistics, trends, and changes from previous years. Some highlights are:

- Submissions to SIGMOD 2018 were down 6% from 2017. The acceptance rate of research papers was 20%, which is in line with recent years.
- Reviewers showed a strong bias to reject borderline papers rather than giving authors the opportunity to revise. By being biased in the second round in favor of offering authors the opportunity to submit a revision, we increased the acceptance rate significantly from the first to second round. We strongly recommend that future PC chairs adopt this bias.
- To help ensure high quality reviews, we gave PC members a light reviewing load and ensured that 95% of review assignments went to PC members who bid Eager or Willing to review the paper. Nevertheless, approximately 20-25% of reviews are unacceptably shallow. We need to do better.
- The main changes in 2018 were (i) to reduce the number of parallel sessions, (ii) include tutorials during the main conference (Tuesday – Thursday), and (iii) return to clustering industry presentations into separate industry sessions rather than grouping them with research papers on the same topic. To enable (i) and (ii), we shortened the standard presentation time to 20 minutes and offered only 10 minutes to 40% of the papers. Anecdotal evidence is that attendees were happy with these changes.

The paper closes with comments about my previous experiences as PC chair for SIGMOD 1979 and VLDB 2002 and with the evolution of PC processes.

1. SUBMISSIONS

Submission statistics are summarized in Table 1. The acceptance rate for research papers has been constant for the past few years at 20%. However, the absolute number has been declining. There was a big jump in the research submissions in 2016, presumably due in part to the attractive location of the conference, San Francisco. Since then, the number has been declining, but is still 11% higher than 2015.

For the 2018 industry track, we reverted from 2017's invitation-only approach back to the tradition of evaluating unsolicited submissions. The acceptance rate was 38%, and there were two invited papers. For the demonstration track, the acceptance rate was 33%. Seven of 14 tutorial submissions were accepted, and two of the seven were asked to merge into a single 3-hour tutorial.

Table 1 Submission-Acceptance Statistics

		2015	2016	2017	2018
Research	Submitted	413	569	489	458
	Accepted	106 (25%)	116 (20%)	96 (19%)	90 (20%)
Industry	Submitted	18	50	0	40
	Accepted	18	21	0	17
	Invited	0	4	4	2
Demo	Submitted	86	126	90	108
	Accepted	30	31	31	36
Tutorial	Submitted	11	24	16	14
	Accepted	4	10	13	6

Figure 1 shows the number of research submissions with a given number of coauthors. Only 1 of 21 papers with one author was accepted. Most submissions had 2-6 coauthors, with acceptance rates of 18% to 22%. The acceptance rate was much higher for papers with 7-9 coauthors and dropped to zero after that.

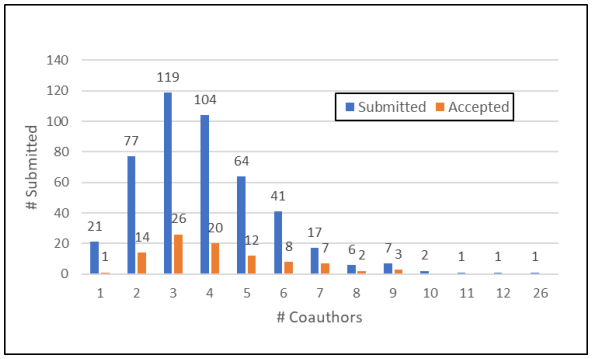


Figure 1 Number of research submissions with a given number of coauthors

As shown in Figure 2, most authors submitted just one research paper. The authors who submitted a lot of papers are professors who were coauthors of many of their students' submissions.

About 20% of submitted abstracts do not result in a submitted paper. In the future, if you are an author of such a paper, please withdraw your paper before or immediately after the submission deadline. This saves work for the PC chair, who has to search for such papers and delete them manually.

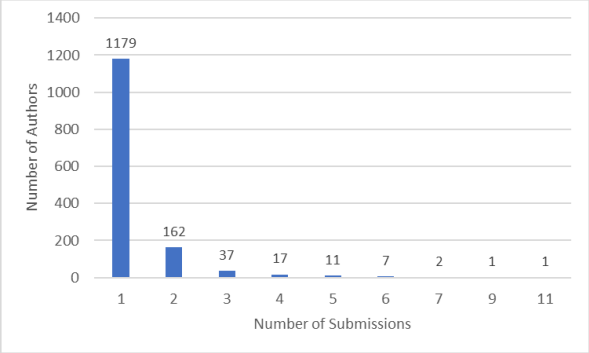


Figure 2 Number of authors with a given number of research submissions

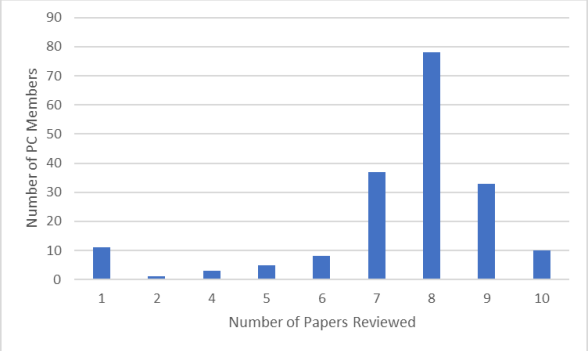


Figure 3 Reviewing Load: Number of PC members with a given reviewing load

2. REVIEWING PROCESS

The quality of the research program depends heavily on the quality of reviewing. It is therefore important to maximize the expertise of reviewers of each paper.

In each of the two submission rounds, PC members and group leaders read abstracts to guide them to bid for papers they wanted to review: Eager, Willing, In-a-Pinch, or Not-Willing. After running the automatic assignment algorithm, I did manual fix-up to improve the result. Overall, 95% of reviewing assignments went to PC members who bid Eager or Willing. (To be precise: 54% Eager, 41% Willing, 4% In-a-Pinch, 0.4% Not-Willing.) All cases of In-a-Pinch or Not-Willing assignments were papers that had an insufficient number of Eager or Willing bids. I manually chose those reviewers.

I assumed that Eager/Willing reviewers would be very knowledgeable about the paper's topic. This didn't always turn out to be true. It can happen because a PC member misjudges the submission's technical focus based on the abstract. I suspect (but cannot prove) that sometimes a PC member will bid Eager/Willing in order to learn about the topic. Please don't do this! It's unfair to the authors and causes extra work for the PC chair, who receives low-confidence reviews and then has to get additional reviews under time pressure.

As in recent years, we used a large PC to ensure a light reviewing load. In each round, most PC members had four papers to review in a 4-week reviewing period, for a total of eight papers for the two rounds. A few had more, either because we had too many submissions in their areas of expertise or because we called on them for a 4th review of a borderline paper (see Figure 3). A few external reviewers reviewed one paper, and a few PC members were recruited for round two to cover topics for which we received more submissions than expected.

Each paper was also assigned to a group leader, who read the paper, guided the discussion after the reviews were in, recommended a decision, and wrote a meta-review. Group leaders also escalated borderline cases to my attention. If reviewers disagreed or they all thought the paper was borderline, we pushed them to discuss it on-line. The average number of comments was the same in 2018 and 2017; see Figure 4 where the number of 2017 submissions was multiplied by 458/489 to normalize for the larger number of submissions than in 2018.

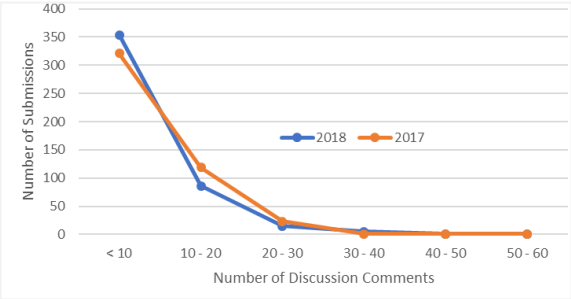


Figure 4 Number of submissions with a given number of discussion comments

3. DECISIONS

Of the 458 research submissions, 20 were desk rejected either because they were very weak or out-of-scope. Of the 210 round-one submissions, we accepted 4 (2%) without revision and asked for a revision of 39 (19%), of which 33 were accepted (85%). I received feedback on the first round from systems researchers that papers they thought were excellent were rejected. The authors speculated this was due to reviews by PC members who weren't systems researchers and didn't know how to evaluate such papers. Since 95% of reviews were by PC members who bid Eager or Willing, I was skeptical of this diagnosis. Therefore, I investigated it by re-reading a lot of reviews and came to a different conclusion: in borderline cases, PC members are strongly inclined to reject rather than offering the authors an opportunity to revise. This inclination seemed to be equally true for all topics, not just for systems papers.

In round one, I tried to mitigate this tendency by encouraging group leaders and PC members to ask for more reviews of borderline papers. Nevertheless, in the common case that all three reviewers believed themselves to be relatively knowledgeable about a paper, they pushed

to reach a decision without additional reviews. They took me up on the offer for only 10% of the round-one submissions. I found many of the 4th reviews helpful. In retrospect, I should have requested many more 4th reviews even if the reviewers and group leader did not. I apologize to authors of rejected papers that I might have been able to save with more effort.

Given my experience in round one, in round two I read the abstract and reviews of most papers. In cases where I thought the reviewers were insufficiently generous, I added discussion comments to try to move the consensus toward a revision. For example, I pushed negative reviewers to be specific about the improvements they would like to see for the paper to be acceptable. I also looked for papers where another review or two might help, and sometimes asked for the reviews even if the PC members and group leader thought it was unnecessary. As a result, we requested revisions of 63 (25%) of the 248 submissions, of which 48 (76%) were accepted. With the 5 papers accepted without revision, that gave us an acceptance rate of 21.4% — significantly higher than round one.

Based on this experience, I recommend that we be more generous in giving authors an opportunity to revise their papers. Often, PC members think that the authors can't make the required changes in the one-month revision period we offered. I believe they should not make this judgement. Authors are highly motivated and will go to great lengths to improve a paper under time pressure, if they have a list of specific improvements that are required. Let *them* decide if they have enough time. I still advocate that we give them a list of specific required changes, and not simply ask the authors to do their best with the reviewers' criticisms, as is often done with journal submissions. However, we should not reject a paper that has a potentially good idea just because the list of changes seems too long.

4. REVIEWING QUALITY

I read hundreds of reviews. Most reviewers do a very good job. They think about the ideas presented in the paper, consider whether the ideas have merit, check that the paper justifies its conclusions, compare the paper's contributions to prior work, and write a review that conveys all this to the authors, the other reviewers, and ideally in enough detail that the PC chair (who probably won't read the paper) can understand it. Great reviews aren't always long, but they are always insightful.

Last year's PC Chair, Dan Suciu, introduced a "Distinguished PC Member Award" to recognize reviewers who did a great job, ~10% of the PC. This year, the vice chairs, group leaders, and I continued this new tradition by recognizing the following PC members: Ashraf Aboulnaga, Manos Athanassoulis, Sebastian Breß, Graham Cormode, Sudipto Das, Khuzaima Daudjee, Aaron Elmore, Ada Fu, Michael Hay, Yuxiong He, Yannis Katsis, Alexandra Meliou, Dan Olteanu, Andrew Pavlo, Peter Pietzuch, Lucian Popa, Semih Salihoglu, Ryan Stutsman, Yufei Tao, & Alexander Thomson.

I regret to report that my impression from reading so many reviews is that 20%-25% of them were unacceptably shallow and sketchy. Previous PC chairs told me this was consistent with their experience. In these cases, the reviewer clearly didn't think hard about the paper, and the review reflects it by offering just a few cheap shots, e.g., the motivation is weak, some sections are hard to understand, more experiments are needed, there are too many typos, and some references are missing. Probably all of that was true, but what did the reviewer think of the ideas? Are they good or bad ideas? Why? What is was weak about the justification for the proposed innovation? What did you expect to see that's lacking?

Weak reviews are not a measure of the strength of the reviewer as a researcher. Many were from people with excellent publication records. A few PC members consistently produced weak reviews. Some weak reviews were from reviewers who also wrote very good reviews.

Given that PC members were asked to review only four papers in each round, the problem cannot be the reviewing load. I suspect the following scenario is common: PC members are busy, and they treat reviewing as their lowest priority task. A PC member reads a paper once. Without thinking hard or spending much more time on it, he or she writes a few obvious criticisms, plus a Weak Accept or Weak Reject rating, depending on the PC member's first impression. Task complete.

Unfortunately, writing a good review is time-consuming. To do it, I usually have to read a paper three times. My first reading is to understand the main ideas and get an overall opinion of the work. On the second pass, I start writing the review while I'm reading, commenting on each section as I go. But that writing activity usually makes me question some of my criticisms, which requires a third pass to sharpen my arguments. The hours usually add up to a full working-day, sometimes more, over several days. Despite the effort, I usually find the time well spent because it forces me to think deeply about a topic, even if the submission is quite weak.

With a 170-person PC, it's a statistical certainty that some reviewers will have an unexpected problem with work, family, or health that prevents them from investing enough time to do a competent review. But that does not account for a quarter of the reviews being unacceptably shallow. We need to improve. Which means that some of you reading this article need to improve.

A weak review leaves a lasting negative impression on other reviewers of the paper. It certainly lowered my opinion of the technical depth of some PC members. I'd have thought that PC members who submit superficial

reviews would be embarrassed when they see their colleagues' substantial reviews. Apparently not.

Moreover, weak reviews are hugely unfair to authors. We all know the weeks, or more often months of effort it takes to write a paper, even a weak one. Authors have a right to expect reviewers to spend enough time to give substantial feedback.

I don't know of any silver bullet to fix this problem. Here are some suggestions I've heard that might encourage people to write better reviews:

1. Have each PC chair ask PC chairs of the last few database conferences to review the proposed list of PC members, to identify reviewers they would not recommend. Not all PC chairs do this, and often only for the preceding conference, so it misses some weak reviewers.
2. Ask authors to vote on whether each review of their paper was well done—thumbs up or down. This isn't a vote on whether the author agrees with the review—only on whether the review reflects a serious attempt at understanding the work and thinking about its novelty, importance, and correctness. We should report it as an aggregate, only to PC members who reviewed enough papers that would make the feedback anonymous.
3. Ask PC members to vote on the quality of other reviews of papers they reviewed. Again, report it as an aggregate only to PC members who reviewed enough papers that would make the feedback anonymous, perhaps merging it with the result of (2).
4. PC chairs and group leaders could ask some PC members to strengthen particular reviews. I did this in a few cases where I thought a gentle push would get the desired effect. In general, it is hard because it is a direct criticism of a PC member's work, which everyone involved finds uncomfortable.

5. THE CONFERENCE PROGRAM

Since 2005, SIGMOD's policy has been to accept all papers that passed the PC's quality bar, with no budget for the maximum number of presentation slots. Then the program schedule was adjusted to give all papers a presentation slot. This has led to a conference schedule of 5-6 parallel SIGMOD sessions, plus a PODS session on Tuesday and Wednesday. Table 2 reports the number of parallel sessions, including industry, tutorial, and demo sessions.

As an experiment, SIGMOD 2016 organizers compressed the program into 3 parallel sessions. To fit all papers into the schedule, they gave each paper a 15-minute slot, which meant about 12 minutes of presentation plus 3 minutes of Q&A, and they moved tutorials to Friday after the conference. Instead of separate industry sessions

Table 2 Number of parallel SIGMOD sessions

Year	Tuesday	Wednesday	Thursday
2009	5	6	5
2010	6	6	6
2011	6	6	6
2012	5	5	5
2013	5	5	6
2014	5	5	6
2015	5	5	6
2016	3	3	3 (tutorials on Friday)
2017	5	5	5 (tutorials on Friday)
2018	4	4	4

sessions, they grouped presentations of industry papers with research papers on the same topic. In 2017, the conference reverted to 5-way parallelism and compensated by introducing plenary teaser talks. It kept the tutorials on Friday and again mixed industry and research papers in the same sessions.

I thought the 2016 experiment was largely successful, as did many attendees who responded to the survey distributed after the conference. However, I found the 15-minute slot too short for many papers, and longer than necessary for others. Therefore, with the approval of the 2017 SIGMOD officers, I decided to try the compressed program again with three modifications: (i) different lengths of presentation slots, 20 minutes or 10 minutes, depending on the paper, (ii) tutorials as a 4th parallel session on Tuesday – Thursday, and (iii) industry presentations placed into separate industry sessions.

There were 54 long presentations and 39 short ones. The decision of long vs. short had several phases. During the reviewing process, PC members were asked to recommend whether each paper, if accepted, should be offered a full presentation slot. Then research PC group leaders made a recommendation for each accepted paper they supervised—definitely 20 minutes, 20 minutes if there's time available, no preference, or definitely 10 minutes—based on reviews, reviewer discussions, and their own judgment, without knowing the identity of authors. Their recommendation is not necessarily a quality metric. They recommended "definitely 10" for some papers highly-rated by reviewers, because the topic was narrow, could be explained in 10 minutes, or couldn't be explained in 20 minutes so extra time wouldn't help. All papers rated definitely-20 or 20-if-there's-time were given long slots, plus some of the no-preference ones. For the latter, the final decision was based on many factors, such as topic diversity, institutional diversity, and the time available in the relevant session.

The industry PC chairs, Sam Madden and Neoklis Polyzotis, were given a free hand in choosing industry

submissions and scheduling them in four sessions. Most of the papers were chosen with the help of the Industry PC. There were also two invited papers. I made the mistake of not asking the industry chairs to align the lengths of talks with the 20- and 10-minute boundaries of research talks. This interfered with session hopping, which didn't occur to me until it was too late to change.

Due to recent changes in U.S. immigration policy, some authors have been unable to attend a conference to present their papers. To accommodate these authors, the SIGMOD officers and VLDB Endowment agreed to allow papers that could not be presented in a SIGMOD or VLDB conference to be presented instead at the next such conference. As a result, three research papers and one demonstration paper from VLDB 2017 were presented at SIGMOD 2018.

Authors of ACM TODS papers can present their paper as a poster at the next SIGMOD conference after the paper's publication. One such TODS paper was presented as a SIGMOD 2018 poster.

I considered compressing the program into three parallel sessions: 2 parallel paper sessions, plus 4 industry sessions, 4 tutorials, 2 demo sessions, and a panel. Using the same schedule as 2018, there would be 960 minutes of presentation time for research papers. With 93 presentations, all of them would have only 10 minutes. To offer 20 minutes for some presentations, some papers would get only a 3-minute teaser talk. I seriously considered doing this for 2018. But when I saw that with three parallel papers sessions we could give everyone at least a 10-minutes talk, I dropped that plan.

At SIGMOD 2017, in place of traditional keynotes, there were invited plenary talks on hot topics by database researchers. In my opinion, the talks were excellent, and the concept of invited database research talks is worth repeating. However, I would make space by reducing the number of long paper presentations and reserve plenary slots for high-profile invited speakers who would not ordinarily attend SIGMOD.

6. PROCEEDINGS AND BOOKLET

The proceedings and booklet handout were prepared in parallel with the conference program. There is manual effort in preparing both of them, which requires great care to avoid inconsistencies. Many authors make last-minute changes to their paper's title and author affiliations, and to schedule constraints that affect session assignment. Maybe someday this will all be generated automatically from a single database. Until that nirvana arrives, authors should do their best to notify the PC and proceedings chairs as early as possible of such changes. They should also strive to submit their non-technical material on time, e.g., photo and bio for tutorial speakers, to assist in the booklet preparation process.

7. HISTORICAL NOTES

This section summarizes my experiences as PC chair of two earlier major database research conference.

7.1 SIGMOD 1979

The last time I was SIGMOD PC Chair was for SIGMOD 1979. For the amusement of younger readers (nearly everyone, I guess) and to capture a bit of history, let me describe what that activity was like. Unfortunately, I no longer have a written record about the PC process, so my foggy memory will have to suffice.

In those days, the Internet was a research project, not a utility. And the World Wide Web was still about 13 years in the future. Therefore, everything was done via hard copy and the postal service. The latter slowed down the process a lot. The schedule had to leave enough time for coast-to-coast mail delays of 4-5 days.

People on the SIGMOD mailing list received a hard-copy call-for-papers in the mail. The call-for-papers was usually published a year in advance, so it could be distributed at the previous SIGMOD conference.

Except for researchers at a few wealthy labs who had access to fancy printers, most authors prepared their submissions with a text editor and impact printer. (My first access to a laser printer came 5 years later.) If the paper had a lot of fancy math, then it might have required using a typewriter (i.e., no computer). Authors had to mail five photocopies of their submission to the PC Chair (i.e., me), ensuring I would receive it before the submission deadline. There were about 75-80 submissions to SIGMOD 1979, the vast majority of which were from U.S. universities and research labs.

I knew each PC member well enough to assign papers that were within their areas of expertise. After doing the reviewer assignments, I mailed a package of papers to each PC member, with copies of the review form for them to fill out. That left two copies of the paper in my file, one for me and one for an extra reviewer if needed.

During the reviewing period, I read all the submissions. I'm told this wasn't common practice for SIGMOD PC chairs, but I doubt I was the first or last to do so. About a quarter of them were so weak that after a half-hour of reading it was obvious they would be rejected, so I didn't have to dig deeper. I read the rest more carefully, but even so, it was a manageable load for a ten- to twelve-week reviewing period.

PC members mailed (mostly hand-written) reviews to me, to arrive before the reviewing deadline, which was a week before the face-to-face PC meeting, which all PC members attended. I produced a list of all the papers in order of their average review score. Unlike today, we had a quota of how many papers to accept, about 24, to fill a two-and-a-half-day single-track program. We started at the top of the list and accepted papers until we

hit a controversial one. Then we switched to the bottom of the list and rejected papers until we hit a controversial one. Most of the discussion happened on the papers in the middle. We talked about them one-by-one until we converged on a final list. Then it was dinner time, the reward for a day of intense discussions.

7.2 VLDB 2002

I was overall PC chair for VLDB 2002. In response to a request by the VLDB Endowment Board to strengthen the trend of broadening the database field beyond database engines, we had two program committees, one for Core Database Technology (chaired by Raghu Ramakrishnan) and another for higher levels of the stack, called Infrastructure for Information Systems (IIS) (chaired by Yannis Ioannidis). The committees accepted 38 of 209 and 31 of 222, respectively (16% overall).

VLDB continued splitting the research PC into two tracks until 2011. By then, it was agreed that the community sufficiently welcomed IIS papers that they no longer needed a separate PC to obtain a fair hearing.

The PC process was similar to SIGMOD 2018, and very different from SIGMOD 1979. Like today, everything was done on-line: submissions, reviews, and reviewer discussions. If I recall correctly, this was the first VLDB that required submissions to be in camera-ready format, to avoid arguments about whether a submission exceeded the length limit. A big difference from today's conference was that we did not offer authors an opportunity to revise a paper and resubmit it for a second evaluation. That is, every paper was accepted or rejected.

At that time, there were growing complaints that PC decisions were too random. As PC chair, I was on the front-line listening to those complaints. As a result, after the conference, I started lobbying to improve the process by having an on-line journal with the same structure as a PC but including a revision cycle. I was not alone in promoting change. Rick Snodgrass, then Editor-in-Chief of ACM TODS, worked to speed up turnaround time to make TODS as appealing to authors as conferences. In 2003, he and I proposed to the VLDB Endowment Board and SIGMOD Executive Committee that borderline rejected papers from one conference could be revised and resubmitted to the next one with the same reviewers, plus one new reviewer for the receiving conference. This process started in 2005 and ran for a couple of years. We also suggested this evolve into an on-line journal, but it was viewed as too radical and rejected. Over the next several years, I presented versions of that concept at CIDR 2003, in panel sessions on PC processes at SIGMOD 2004 [1] and VLDB 2005 [2], and at annual VLDB Endowment Board meetings in 2003-2005. There were many other proposals, some presented in [1] and [2] and some discussed privately at SIGMOD and VLDB Board meetings. I was insufficiently persuasive to get either organization to agree to the change. However, after I rolled off the VLDB Endowment Board in 2006, H.V. Jagadish got approval for a related proposal: changing VLDB to an on-line journal, PVLDB, with monthly submissions year-round. I believe the approval was helped by his agreement to serve as its first editor-in-chief, something I was not willing to do. There is widespread agreement that PVLDB has been a big success, which was one of many contributions for which Jagadish received the 2013 SIGMOD Contributions award.

An aside: The VLDB 2002 general chair, Fred Lochovsky, and I pushed for approval to publish the proceedings only in electronic form. The VLDB Endowment Board declined our request and insisted on a printed copy, which ended up as a weighty tome of 1050 pages. Old habits die hard. It took a few more years before hard-copy proceedings were abandoned.

8. FINAL REMARK

Our community has been at the forefront of changes in the PC conference reviewing process for many years. Today's processes are imperfect, and we should continue to strive to improve them. If I could wave a magic wand to get only one improvement, it would be that all PC members invest enough time to give substantial thought to every paper they review and write a detailed evaluation. That would go a long way to increase author satisfaction of the processes that we currently use.

9. ACKNOWLEDGMENTS

Assembling the SIGMOD 2018 program was the work of over 200 people: PC members, group leaders, and track chairs. I'm enormously grateful for their help. I would especially like to thank the two PC vice-chairs, Luna Dong and Mohamed Mokbel, who did a huge amount of work, collaborating closely with me on all aspects of the program. They deserve a lot of credit for the best aspects of the program.

10. REFERENCES

[1] Michael J. Franklin, Jennifer Widom, Gerhard Weikum, Philip A. Bernstein, Alon Y. Halevy, David J. DeWitt, Anastasia Ailamaki, Zachary G. Ives: "Rethinking the Conference Reviewing Process - Panel." SIGMOD Conference 2004: 957

[2] Philip A. Bernstein, David J. DeWitt, Andreas Heuer, Zachary G. Ives, Christian S. Jensen, Holger Meyer, M. Tamer Özsu, Richard T. Snodgrass, Kyu-Young Whang, Jennifer Widom: "Database Publication Practices." VLDB 2005: 1241-1246.

Report on the First International Workshop on Incremental Re-computation: Provenance and Beyond

Paolo Missier
School of Computing,
Newcastle University, UK
paolo.missier@ncl.ac.uk

Tanu Malik
School of Computing
DePaul University
tmalik1@depaul.edu

Jacek Cala
School of Computing,
Newcastle University, UK
jacek.cala@ncl.ac.uk

1. INTRODUCTION

In the last decade, advances in computing have deeply transformed data processing. Increasingly systems aim to process massive amounts of data efficiently, often with fast response times that are typically characterised by the 4V's, i.e., Volume, Variety, Velocity, and Veracity. While fast data processing is desirable, it is also often the case that the outcomes of computationally expensive processes become obsolete over time, due to changes in inputs, reference datasets, tools, libraries, and deployment environment. Given massive data processing, such changes must be carefully accounted for, and their impact on original computation assessed, to determine how much re-computation is needed in response to changes.

A core challenge is how to optimise re-computation in the presence of changes, given an existing process execution baseline. Specific research questions include (1) how, and under what assumptions, can re-computation be optimised using incremental and/or partial processing techniques given the baseline, and (2) how do we determine the impact of a set of changes on the outcomes, in order to decide when changes should trigger re-computations.

In this article we report on the proceedings of the *First International Workshop on Incremental Re-computation: Provenance and Beyond* (IRPb), which was organised to explore the breadth and depth of the re-computation problem, with specific emphasis on the role of provenance in this area. Within this scope, the workshop provided a forum for experts to constructively explore theoretical, systems-oriented, and provenance-related challenges in developing and using incremental re-computation based systems.

IRPb was held in conjunction with ProvenanceWeek 2018, a bi-annual week-long event that includes the 7^{th} edition of the International Provenance and Annotation Workshop (IPAW), and the 10^{th} Usenix Workshop on the Theory and Practice Of Provenance (TAPP). The format chosen for the workshop was designed to encourage discussion without requiring a paper contribution, other than an abstract. Held over two half-days, IRPb consisted of a collection of 12 short talks (15-20') plus ample time for discussion, given by recognised experts in the areas within the scope, and 2 longer keynote talks. Abstracts and presentations are available at `https://tinyurl.com/y7c8vttn`.

2. WORKSHOP TOPICS

With each of the 14 contributors presenting their own perspective on the topic, we have used the following categories to characterise the contributions, using tags to annotate the individual talks.

Re-computation, i.e., the repeating of a process execution, all or in parts, under slightly different inputs or configuration each time, and making use of one or more prior execution baselines as a basis for optimization. We use tags **#howto-recomp** and **#using-recomp** to distinguish research that describes techniques that advance the state of the art on performing re-computation, from research that makes use of such techniques, respectively.

Incremental computation. This is naturally viewed as one of the ways re-computation can be optimised, however it is arguably more general, as it does not require a prior baseline (first time executions may be incremental). As before, we use tags **#howto-incr-comp** and **#using-incr-comp**.

Approximate computation, a well-established field is identified using tag **#howto-approx-comp**.

Provenance, including all phases of its lifecycle, namely capture, storage, query, and analysis. Again, we make a distinction between **#using-provenance** and **#for-provenance**.

Contributions also covered a diversity of applications areas, ranging from the Life Sciences (genomics and metagenomics), machine learning, data journalism, transportation science, and large-scale

simulations, as well as research areas ranging from databases and data integration, programming languages, reproducibility of e-science processes (scientific workflows, including workflow steering), process mining, and naturally, core provenance research.

3. CONTRIBUTIONS

3.1 *Keynote:* Language-based issues in incremental computation

James Cheney's talk presented the different forms on incremental computation and their use in programming languages. He outlined work (mostly by others) in three sub-areas (i) on incremental, (ii) self-adjusting and (iii) bidirectional computation, clarifying the meaning and subtle formalism difference within each area. James presented a simplified example of a delta data structure used to compute the square of the sum of two numbers, and described how such data structure is maintained using static differentation and more involved incremental lambda calculus as described in the seminal paper of Cai et. al. [2]. Self-adjusting computation implies recomputing efficiently as the input is changed, using caching to avoid recomputation of sub-expressions whose results have not changed. James covered work from Acar et.al [4] and described the primary idea of self-adjusting traces, which first use execution to generate annotations (the trace), and then use the annotated program trace to make subsequent runs probably faster compared to running "from scratch". He highlighted the opportunity of including provenance-like information in these traces to improve incremental computation. The limitation of the approach is that generating annotations often requires slight program modification, but also mentioned significant progress on how to reduce the annotations. Finally, he covered bidirectional computation, which means updating the input to a computation to be consistent with a proposed new output: a generalization of the view update problem. In this he presented a recent contribution to incremental view update problem in relational databases, which is of interest for computing "missing answer" or hypothetical explanations for database settings.

3.2 *Keynote:* Modern Dataflow

Frank McSherry's keynote talk summarized his recent work on Differential Dataflow [11], a data-parallel programming and execution model for scalable and incremental computation, which is based on the 'Timely Dataflow' framework [12] and a data serialization library called 'Abomonation'.

Differential Dataflow is a collection-oriented programming model in which users program their algorithms with a set of functional operators, while the system manages incremental changes to the input data. The set of operators includes well-known data-parallel functions like: map, reduce and join but also iterate, which allows *incremental and iterative* algorithms to be implemented. The key element of the model, which makes it distinct from other approaches to incremental computation, is that in Differential Dataflow the state of computation and its updates are associated with a multi-dimensional logical timestamp. Such association allows the system to maintain a partially ordered set of versions (data updates) rather than only the most up-to-date coalesced state of computation.

Frank compared the efficiency of his platform and showed at least an order of magnitude speed-up for graph-based problems like PageRank and connected components. He also illustrated computation on directed acyclic graphs in which 40% of the computation does not change the output due to change in inputs, whereas 60% changes it only moderately.

The talk also briefly introduced two other elements of the Modern Dataflow platform: the Timely Dataflow framework, to scale the same program up from a single thread to distributed execution on a cluster of machines, and the Abomonation library for fast data serialization, in the Rust language.

3.3 Short Presentations

Answering Why-Not queries Against Scientific Workflow Provenance Khalid Belhajjame (University Paris-Dauphine, France) focused on a variation of the well-known problem of answering *Why-not* database queries, that is, to explain why a certain tuple is not returned as part of a query answer. The problem finds a similar formulation but requires a new approach when the question is posed on the result of a workflow execution. The proposed approach is shown to require the re-computation of parts of the workflow. **#using-recomp**, **#for-provenance**.

The Marriage of Incremental and Approximate Computing Pramod Bhatotia (University of Edinburgh) described differences between incremental and approximate computation. In essence, both paradigms rely on computing over a subset of data items instead of computing over the entire dataset, but they differ in their means for skipping parts of the computation. Pramod suggests that the two approaches are complementary and can be com-

bined, namely by using a stratified sampling algorithm that biases the sample selection to the memoized data items from previous runs. The resulting implementation, based on Apache Spark Streaming, is part of a data analytics system called *IncApprox* (Incremental + Approximate Computing) [9]. **#howto-incr-comp**, **#howto-approx-comp**.

Supporting Incremental Re-Computation with Whole System Provenance: Issues and Approaches Ashish Gehani (SRI International, USA) focused on *whole system provenance*, that is, provenance collected from observations of system-level events during process execution, as a way to make process re-computation efficient, i.e., by reducing the fraction of computation that needs to be performed again. The talk addressed practical issues that arise in this context, and outlined approaches to address them. The challenges include ephemeral intermediate artifacts, conflated causality, dynamic runtime environments, and external dependencies. **#howto-recomp**, **#using-provenance**

TensorCell - approximating outcomes of computer simulations using machine learning algorithms. Pawel Gora (University of Warsaw, Poland) presented an approach for approximating the result of traffic simulations using neural networks. Simulating the collective behaviour of traffic lights in a large city, such as Warsaw, is computationally complex. Exploring "what-if" scenarios by repeating the simulation becomes prohibitive. The authors tested the hypothesis that reasonable approximations of the expected waiting time at each of the lights can be obtained by training a neural network on an exemplar set of light configurations (on 15 significant crossroads). The results are encouraging, with average prediction error varying between 1.18% and 6.8%, with sub-second processing time on the network. **#howto-approx-comp**.

Progressive Provenance Capture Through Re-computation Paul Groth (Elsevier Labs, NL) described his research on using record-replay technology within virtual machines to incrementally add additional provenance instrumentation by replaying computations after the fact. **#using-Recomp**, **#for-provenance**

Incremental Recomputation in Data Integration Melanie Herschel's (University of Stuttgart, Germany) contribution focused on complex data integration processing pipelines, which are often developed and maintained incrementally, and require multiple iterations to reach satisfactory results. The talk discussed the potential benefits of how-provenance and what-if analysis in supporting incremental re-computation. **#using-incr-comp**, **#using-provenance**

Incremental Recomputation: Those who cannot remember the past are condemned to recompute it Bertram Ludascher (University of Illinois at Urbana Champaign, USA) explored the connection between re-computation, Models of Computation (MoC), and *models of provenance* (MoP). He made the point that "computing with deltas" has been common for a variety of "Models of Computation", from Datalog (Delta Computations and semi-naive evaluation, Statelog (Stateful Datalog)) to incremental view maintenance in databases, to workflow programming. In particular, a connection has been made over the years between MoCs associated with workflows, for example when using Kepler [10], and corresponding MoPs [1], and can also serve as a foundation for implementing provenance-based fault tolerance mechanisms [8], i.e., using checkpoints and partial re-run. **#howto-incr-comp**, **#howto-recomp**

Incremental Recomputation in Containers Tanu Malik (DePaul University, USA) described the need for incremental computation in reproducible computation. She showcased Sciunit (`https://sciunit.run`) reproducible containers, which capture necessary and sufficient binaries and data so as to repeat the computation in a new environment, but must be entirely re-evaluated every time a change to input parameter or dataset is made [13]. She highlighted the need of incremental re-computation techniques on versioned provenance graphs. **#howto-recomp**, **#using-recomp**.

Collecting Provenance of Steering Actions Mattoso and Sousa (UFRJ, Brasil) presented *workflow steering* [14], a form of human-in-the-loop scientific workflows where experts are given the chance to repeatedly tune some of the process' parameters at runtime, with the aim to significantly improve execution performance or improve result quality. The problem addressed in the talk is how to track and record users' steering actions, i.e., using a provenance-based framework. **#howto-incr-comp**, **#using-provenance**.

Provenance and recomputing in the realm of large scale environmental sequence analysis Folker Meyer (Argonne National Labs, USA) talked about the needs for re-computing in the realm of large scale environmental sequence (metagenomics) analysis, where frequent changes in the underlying knowledge databases render in-silico analy-

sis results both uncertain and unstable. Taking the perspective of a large-scale analysis provider (*MG-RAST*, `https://mg-rast.org`) which caters to tens of thousands of scientists from many domains, the talk highlighted the need for effective re-computation tools. **#using-recomp**, **#using-provenance**.

The ReComp project: an overview Paolo Missier (Newcastle University, UK) started from the observation that the outcomes of computationally intensive processes (data processing pipelines, simulations) are often time-sensitive, as they depend on algorithms, tools, and reference databases that evolve over time. A re-computation problem naturally occurs when some of the changes in these elements invalidate some of the outcomes. The problem is to estimate which of the past outcomes are affected by a change, and to what extent. The talk provided an overview of *ReComp* (`http://recomp.org.uk`), a generic framework designed to determine the minimal process fragment that requires re-computation [3]. **#howto-recomp**, **#using-recomp**.

Handling late data in process mining algorithms Tomasz Pawlowski and Jacek Sroka (University of Warsaw, Poland) situate their research at the intersection of process mining, a very mature research area, and stream data processing. They observe that, while a number of process mining techniques exist analyse and visualise repetitive processes, those mostly operate offline, on static event logs. They focus instead on online analysis of streams of logs, where problems occur when some events are logged out-of-order. For instance, out-of-order events happen when a streaming process is offline. The authors investigate how in this setting incremental re-computation of existing process mining algorithms occurs—in particular to handle out-of-order data without repeating the whole data mining computation from scratch. **#howto-incr-comp**.

Self-Explaining Computation with Explicit Change Perera (University of Edinburgh, UK) proposed a notion of *self-explaining computation with explicit change*. He used Jupyter notebooks as an example of *data-driven storytelling* that can offer some form of *explorable explanations*. But Prera mentioned there are limitations with respect to transparency of explainations. The goal of self-explaining computation with explicit change is to increase transparency by making it explicit how parts of a computation relate to other parts, and how changes cause other changes. The latter point connects this research with incremental (re-)computation, with the idea to leveraging ideas from partial and incremental computation as well as self-explaining computation [4] [6, 7] to user interfaces, namely by enabling components which use provenance [5] to support slicing and delta-visualisation, making explanations accessible directly from data views. **#using-incr-comp**

4. REFERENCES

[1] Anand, M. K., Bowers, S., McPhillips, T. M., and Ludäscher, B. Exploring Scientific Workflow Provenance Using Hybrid Queries over Nested Data and Lineage Graphs. In *SSDBM* (2009), pp. 237–254.

[2] Cai, Y., Giarrusso, P. G., Rendel, T., and Ostermann, K. A theory of changes for higher-order languages: Incrementalizing λ-calculi by static differentiation. In *ACM SIGPLAN Notices* (2014), vol. 49, ACM, pp. 145–155.

[3] Cała, J., and Missier, P. Selective and Recurring Re-computation of Big Data Analytics Tasks: Insights from a Genomics Case Study. *Big Data Research in press* (aug 2018).

[4] Cheney, J., Acar, U. A., and Perera, R. *Toward a Theory of Self-explaining Computation*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 193–216.

[5] Cheney, J., Chiticariu, L., and Tan, W.-C. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases 1*, 4 (2009), 379–474.

[6] Griffin, T., Libkin, L., and Trickey, H. An improved algorithm for the incremental recomputation of active relational expressions. *IEEE Transactions on Knowledge and Data Engineering* (1997).

[7] Horn, R., Perera, R., and Cheney, J. Incremental relational lenses. *Proc. ACM Program. Lang. 2*, ICFP (July 2018), 74:1–74:30.

[8] Köhler, S., Riddle, S., Zinn, D., McPhillips, T., and Ludäscher, B. Improving workflow fault tolerance through provenance-based recovery. In *SSDBM* (2011), pp. 207–224.

[9] Krishnan, D. R., Quoc, D. L., Bhatotia, P., Fetzer, C., and Rodrigues, R. Incapprox: A data analytics system for incremental approximate computing. In *25th International Conference on World Wide Web* (2016), pp. 1133–1144.

[10] Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E. A., Tao, J., and Zhao, Y. Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice and Experience 18*, 10 (2005), 1039–1065.

[11] McSherry, F., Murray, D. G., Isaacs, R., and Isard, M. Differential dataflow. In *CIDR* (2013).

[12] Murray, D. G., McSherry, F., Isaacs, R., Isard, M., Barham, P., and Abadi, M. Naiad: a timely dataflow system. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (2013), ACM, pp. 439–455.

[13] Pham, Q., Malik, T., and Foster, I. Using Provenance for Repeatability. In *TaPP* (2013).

[14] Souza, R., and Mattoso, M. Provenance of dynamic adaptations in user-steered dataflows. In *IPAW* (2018), pp. 16–29.