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, Rada Chirkova, 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, and Tim Kraska

SIGMOD Information Director:
Curtis Dyreson, Utah State University

Associate Information Directors:
Huiping Cao, Georgia Koutrika, Wim Martens, and Sourav S Bhowmick

SIGMOD Record Editor-in-Chief:
Rada Chirkova, NC State University

SIGMOD Record Associate Editors:
Azza Abouzied, Lyublena Antova, Vanessa Braganholo, Aaron J. Elmore, Wim Martens, Kyriakos Mouratidis, Dan Olteanu, Divesh Srivastava, Pınar Tözün, Immanuel Trummer, Yannis Velegrakis, Marianne Winslett, and Jun Yang

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

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

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

SIGMOD Awards Committee:
Martin Kersten (Chair), Surajit Chadhuri, David DeWitt, Sunita Sarawagi, and Michael 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, and Pinar Tozun

SIGMOD Systems Award Committee:
Michael Cafarella (Chair), Michael Carey, David DeWitt, Yanlei Diao, Paul Larson, and Gustavo Alonso

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)
Anastasia Ailamaki (2019)

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, and Utkarsh Srivastava (2018); Xiaofeng Bao, Charlie Bell, Murali Brahmadesam, James Corey, Neal Fachan, Raju Gulabani, Anurag Gupta, Kamal Gupta, James Hamilton, Andy Jassy, Tengiz Kharatishvili, Sailesh Krishnamurthy, Yan Leshinsky, Lon Lundgren, Pradeep Madhavarapu, Sandor Maurice, Grant McAlister, Sam McKelvie, Raman Mittal, Debanjan Saha, Swami Sivasubramanian, Stefano Stefani, and Alex Verbitski (2019)

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)
Ahmed Elmagarmid (2019)

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
- **2019** *Winne*r: Joy Arulraj. *Honorable Mention*: Bas Ketsman

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

[Last updated: December 31, 2019]

Editor's Notes

Welcome to the December 2019 issue of the ACM SIGMOD Record!

This issue starts with the Database Principles column featuring an article by Cuenca Grau, Horrocks, Kaminski, Kostylev, and Motik that discusses Limit Datalog, query language that could be appropriate for data-analysis applications. The language extends the well-known declarative language Datalog in ways that make the result powerful enough to naturally capture important data-analysis tasks on complex data sets, while keeping the reasoning no harder than in the case of Datalog. The importance of the language being declarative is that in specifying queries, data analysts can focus on the goals to be achieved in the analysis, rather than on the procedural implementations of the needed tasks. The article details the construction and complexity considerations in designing the language, and supplies illustrations via examples motivated by practical applications. The authors also discuss how the language can be used as a basis for understanding the expressive power of key data-analysis constructs, and outline directions of future work.

The Surveys column features an article by Zhang, Zhang, Wu, Johns, and He that studies the problem of improving the utilization of modern hardware in data-stream processing systems (DSPSs). The article discusses the challenges of optimizing system latency and throughput toward achieving real-time data analytics on large-scale streaming data, in presence of underutilization of the available hardware resources. Toward addressing the challenges, the authors present a systematic study of recent works in the field, organized along the axes of computational optimization, stream I/O optimization, and query deployment. The article also summarizes how hardware-conscious optimization techniques mitigate the gap between DSPSs and the requirements of real-time stream processing that were formulated by Stonebraker and colleagues in their 2005 SIGMOD Record paper. The article concludes by formulating major open issues and proposing directions of future research in the area.

The Research Articles column presents an article by Papadakis, Tsekouras, Thanos, Giannakopoulos, Palpanas, and Koubarakis that introduces the Java gEneric Data Integration (JedAI) toolkit. The focus of the toolkit is on addressing two major challenges in end-to-end entity resolution: development of extensible open-source tools and provision of solutions that apply not only to structured, but also to semi- or even unstructured data. The article details the composition and functionalities of JedAI, outlines its user-friendly interface, and reports the results of the experimental performance comparisons of the toolkit with three state-of-the-art packages described in the literature, using six real data sets. The JedAI code is available from the authors of the article.

The Distinguished Profiles column features Anastasia Ailamaki, professor at the Swiss Federal Institute of Technology (EPFL), and former professor at Carnegie Mellon. Natassa is an ACM Fellow and a Sloan Fellow; she has received the European Young Investigator Award, the 2018 Nemitsas Prize in Computer Science from the President of the Republic of Cyprus, and the 2019 Edgar F. Codd Innovation Award from ACM SIGMOD, as well as ten Best Paper awards for her research papers. Her Ph.D. is from the University of Wisconsin, Madison. In this interview, Natassa talks about her research in two very different systems areas, about merit-based hiring, and about how she approaches working with students and staff in her large research group. She also discusses challenges and tradeoffs in hardware/software codesign, including potential opportunities for both database researchers and database users. Natassa outlines ideas for emerging applications, such as just-in-time access to data, and shares her views on research in energy efficiency for database-management

systems. She also provides insights on a range of other topics, including the additional things that she could work on if she had the time, and gives advice on database research and mentorship.

This issue features two reports. First, it is our pleasure to include the 2018 Seattle Report on database research. Every few years a group of database researchers meets to discuss the state of database research, its impact in practice, and important new directions. This report summarizes the discussion and conclusions of the ninth such meeting, held October 9-10, 2018 in Seattle. The meeting identified the changes that have taken place in the last five plus years, for reasons that include recent technological breakthroughs in a number of areas, including machine learning and artificial intelligence, the rise of interdisciplinary data science, and new trends in the hardware landscape. These changes have given rise to an unprecedented opportunity for the database-research community to have a transformative impact on today's world. The report details specific research challenges and opportunities in the areas of data science, data governance, cloud services, and database engines. The article also outlines community challenges, including those particular to end-to-end solutions in the hands of users, the data-science software ecosystem, and the impact on university campuses stemming from the rise of data science on the curricula. The second article in the Reports column, by Manghi, Candela, Lazzeri, and Silvello, focuses on the Italian Research Conference on Digital Libraries (IRCDL), annual Italian forum for research topics revolving around digital libraries. The 2019 IRCDL conference took place in Pisa; its theme was "Digital Libraries: Supporting Open Science." Science is increasingly becoming digital, and research results are no longer just traditional scientific publications, but are represented by digital artifacts as well, including data sets, software, and experiments. In this landscape, Digital Libraries are an integral part of the evolution of the research outputs. The 2019 IRCDL conference focused on the topics related to findability, preservation, interlinking, and reuse of the research products via Digital Libraries. The article summarizes the ideas presented and discussed in the conference, and outlines key observations and emerging research directions.

On behalf of the SIGMOD Record Editorial board, I hope that you enjoy reading the December 2019 issue of the SIGMOD Record!

Your submissions to the SIGMOD Record are welcome via the submission site:
https://mc.manuscriptcentral.com/sigmodrecord

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

Rada Chirkova

December 2019

Past SIGMOD Record Editors:

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

Limit Datalog: A Declarative Query Language for Data Analysis

Bernardo Cuenca Grau
University of Oxford

Ian Horrocks
University of Oxford

Mark Kaminski
University of Oxford

Egor V. Kostylev
University of Oxford

Boris Motik
University of Oxford

ABSTRACT

Motivated by applications in declarative data analysis, we study $Datalog_{\mathbb{Z}}$—an extension of Datalog with stratified negation and arithmetics over integers. Reasoning in this language is undecidable, so we present a fragment, called limit $Datalog_{\mathbb{Z}}$, that is powerful enough to naturally capture many important data analysis tasks. In limit $Datalog_{\mathbb{Z}}$, all intensional predicates with a numeric argument are *limit predicates* that keep only the maximal or minimal bounds on numeric values. Reasoning in limit $Datalog_{\mathbb{Z}}$ is decidable if multiplication is used in a way that satisfies our *linearity condition*. Moreover, fact entailment in limit-linear $Datalog_{\mathbb{Z}}$ is Δ_2^{EXP}-complete in combined and Δ_2^{P}-complete in data complexity, and it drops to coNEXP and coNP, respectively, if only (semi-)positive programs are considered. We also propose an additional stability requirement, for which the complexity drops to EXP and P, matching the bounds for usual Datalog. Limit $Datalog_{\mathbb{Z}}$ thus provides us with a unified logical framework for declarative data analysis and can be used as a basis for understanding the expressive power of the key data analysis constructs.

1. INTRODUCTION

Analysing complex datasets is a hot topic in information systems. The term 'data analysis' covers a broad range of tasks such as data aggregation, property verification, and query answering. Such tasks are usually realised in practice using imperative programming languages. However, there has recently been considerable interest in declarative solutions, where the definition of the task is clearly separated from its implementation [1, 18, 25, 26, 29]. The main idea is that users should describe *what* the desired output is, rather than *how* to compute it. For example, instead of computing shortest paths in a graph using a concrete algorithm, one first describes what a path length is and then selects the paths of minimum length. Such a specification is independent of evaluation details, allowing analysts to focus on their task at hand. An evaluation strategy can be selected independently, typically by reusing efficient general algorithms 'for free'.

An essential ingredient of declarative data analysis is a suitable logic-based language for representing the relevant analysis tasks. Datalog [8] is a prime candidate due to its support for recursion, which is needed to express common analysis problems such as shortest path. However, in addition to standard features of Datalog, even basic data analysis tasks often require integer arithmetic or aggregation to capture quantitative aspects of data (e.g., the length of a shortest path). Research on extending recursive rule languages with means for capturing numeric computations dates back to the '90s [3, 6, 12, 16, 20, 22, 28], and is currently experiencing a revival [11, 19, 31]. This large body of work, however, focuses primarily on integrating recursion with arithmetic and aggregation in a coherent semantic framework, where technical difficulties arise due to nonmonotonicity. Surprisingly, little is known about the computational properties of languages integrating recursion with arithmetic, other than the fact that a straightforward combination is undecidable [8]. This result applies also to the languages of existing Datalog-based tools such as BOOM [1], DeALS [30], LogicBlox [2], Myria [29], SociaLite [25], Overlog [17], Dyna [9], and Yedalog [4].

The aim of this article is to lay the foundation for Datalog-based declarative data analysis by developing new extensions of Datalog that are powerful and flexible enough to naturally capture many important analysis tasks, and that yet exhibit favourable computational properties of reasoning. These languages can provide a formal basis for the development of reasoning engines that support complex analytical tasks and provide correctness, robustness, scalability, and extensibility guarantees. They can

also serve as a unified logical framework providing a basis for understanding the expressive power of the key data analysis constructs.

We take as a starting point $Datalog_{\mathbb{Z}}$—a well-known extension of Datalog with stratified negation as failure, integer arithmetic, and comparisons. After reviewing basic definitions in Section 2, in Section 3.1 we present *limit* $Datalog_{\mathbb{Z}}$, which can be equivalently seen as either a semantic or a syntactic restriction of $Datalog_{\mathbb{Z}}$. In limit $Datalog_{\mathbb{Z}}$, all intensional predicates with numeric arguments are *limit* predicates that keep maximal or minimal bounds on the numeric value for each tuple of the other arguments. For example, if we encode a directed graph with edge lengths using a ternary predicate E, then Rules (1) and (2), where dst is a min limit predicate, compute the length of a shortest path from a source node a_s to each node in the graph.

$$\rightarrow dst(a_s, 0) \tag{1}$$

$$dst(x, m) \wedge E(x, y, n) \rightarrow dst(y, m + n) \tag{2}$$

Rule (2) intuitively says that, if x is reachable from a_s with length at most m and (x, y) is an edge of length n, then y is reachable from a_s with length at most $m + n$. If these rules and a dataset entail $dst(a, \ell)$, then the length of a shortest path from a_s to a is at most ℓ; thus, $dst(a, k)$ holds for each $k \geq \ell$ since the length of a shortest path is also at most k. This is different from ordinary $Datalog_{\mathbb{Z}}$, where $dst(a, \ell)$ and $dst(a, k)$ are not semantically connected. In Section 3.2, we show that fact entailment remains undecidable for limit $Datalog_{\mathbb{Z}}$ programs. To ensure decidability, we introduce *limit-linear* $Datalog_{\mathbb{Z}}$, which disallows multiplication of numeric variables that are used in the same stratum. In Section 3.3, we present several examples that show how limit-linear $Datalog_{\mathbb{Z}}$ can capture many practically relevant data analysis tasks.

In Section 4, we establish decidability of fact entailment for limit-linear $Datalog_{\mathbb{Z}}$ and design worst-case optimal algorithms for positive (i.e., negation-free), semi-positive (i.e., with negation only in front of extensional atoms), and arbitrary (i.e., with stratified negation) limit-linear programs. Our results are obtained by a reduction to the evaluation problem for sentences of a specific shape in Presburger arithmetic. In particular, in Section 4.1 we first design a fact entailment algorithm for positive limit-linear programs with coNEXP and coNP upper complexity bounds in combined and data complexity, respectively, and then show that these bounds are worst-case optimal. In Section 4.2, we first show that fact entailment for semi-positive programs can be reduced in polynomial time to the positive case and then design a fact entailment algorithm for arbitrary limit-linear programs that materialises the input stratum by stratum, by relying at each stage on an oracle computing the materialisation of a semi-positive program corresponding to the previous strata. This algorithm provides Δ_2^{EXP} and Δ_2^{P} upper complexity bounds, and we show that these bounds are also worst-case optimal.

The results of Section 4 establish intractability of reasoning over limit-linear programs. In Section 5, we identify fragments of our language for which reasoning is tractable in data complexity, and which are therefore well-suited for data-intensive applications. In particular, using the idea of *cyclic dependency* detection, in Section 5.1 we introduce *stable* programs that allow reasoning to become EXP-complete in combined complexity and P-complete in data complexity (i.e., no harder than for ordinary Datalog). Stability, however, is a semantic condition that is hard to check; thus, in Section 5.2, we identify a syntactic *type-consistency* condition, which implies stability and can be easily checked rule by rule. We then argue that all analysis tasks discussed in our examples can be captured using type-consistent $Datalog_{\mathbb{Z}}$ programs.

Finally, in Section 6, we compare our language with the formalisms underpinning several existing rule-based systems for data analysis.

This paper summarises the results reported in two conference publications [14, 15], and we refer to them for further details.

2. DATALOG$_{\mathbb{Z}}$

We first recapitulate the well-known definition of Datalog with stratified negation and arithmetic over the set of integers $\mathbb{Z}$, which we call $Datalog_{\mathbb{Z}}$. Our formalism is standard and closely related to constraint logic programming (CLP) over the structure $(\mathbb{Z}, \leq, <, +, -, \times, 0, \pm 1, \pm 2, \dots)$ [7, 8].

We assume countably infinite and mutually disjoint sets of *objects*, *object variables*, *numeric variables*, and *predicates*. Each predicate has a nonnegative integer *arity*, and each position of a predicate is of *object* or *numeric sort*. We call predicates $\leq$ and $<$ with two numeric positions *comparison* predicates, and we call all remaining predicates *standard*. An *object term* is an object or an object variable. A *numeric term* is an integer, a numeric variable, or an expression of the form $s_1 + s_2$, $s_1 - s_2$ or $s_1 \times s_2$, where s_1 and s_2 are numeric terms, and $+$, $-$ and $\times$ are the usual *arithmetic functions*. A *constant* is an object or an integer. A *standard atom* is an expression of the form $A(t_1, \dots, t_v)$, where A is a standard predicate of arity v and each t_i is a term of

the sort of position i in A. A *comparison atom* is an expression of the form $(s_1 \leq s_2)$ or $(s_1 < s_2)$, where s_1 and s_2 are numeric terms. We use $(s_1 \geq s_2)$ for $(s_2 \leq s_1)$, $(s_1 \doteq s_2)$ for $(s_1 \leq s_2) \wedge (s_2 \leq s_1)$, and so on. A *positive literal* is a standard or a comparison atom, a *negative literal* is an expression of the form $\mathsf{not}\,\alpha$ for α a standard atom, and a *literal* is a positive or a negative literal.

A *rule* ρ is of the form $\varphi \rightarrow \alpha$, where the *head* α of ρ is a standard atom and the *body* φ of ρ is a conjunction of literals. We consider only *safe* rules, where each variable occurs in a positive body literal. A *fact* is a rule with the empty body and in which all terms are constants (i.e., it mentions neither variables nor arithmetic functions); we usually omit '$\rightarrow$' in facts. A *dataset* is a finite set of facts.

We use the usual *stratification* condition [8] to ensure that negation is 'well-behaved'. A finite set $\mathcal{P}$ of rules is *stratifiable* if it can be partitioned into disjoint subsets $\mathcal{P}[1], \ldots, \mathcal{P}[h]$ called *strata* such that, for each $i \in [1, h]$, each predicate occurring in $\mathcal{P}[i]$ does not occur in the head of a rule in any $\mathcal{P}[j]$ with $j > i$, and each predicate occurring in a negative body literal of a rule in $\mathcal{P}[i]$ does not also occur in the head of a rule in $\mathcal{P}[i]$. When such a stratification exists, we say that $\mathcal{P}$ *admits* h strata.

A ($Datalog_{\mathbb{Z}}$) *program* $\mathcal{P}$ is a finite stratifiable set of rules. A standard predicate A is *intensional* (*IDB*) in $\mathcal{P}$ if it occurs in $\mathcal{P}$ in the head of a rule that is not a fact; otherwise, A is *extensional* (*EDB*) in $\mathcal{P}$. Program $\mathcal{P}$ is *positive* if it does not use negative literals (so $\mathcal{P}$ admits a single stratum), and it is *semi-positive* if the predicate of each negative literal is EDB in $\mathcal{P}$ (thus, $\mathcal{P}$ admits two strata).

We discuss the semantics of $Datalog_{\mathbb{Z}}$ only informally as it is the same as for usual Datalog with stratified negation [8]. An *interpretation* $\mathcal{I}$ is a (not necessarily finite) set of facts. If all the rules of a program $\mathcal{P}$ are satisfied in $\mathcal{I}$ (under the usual semantics of first-order logic with integer arithmetic, assuming that all variables in rules are universally quantified), then $\mathcal{I}$ is a *model* of $\mathcal{P}$ and we write $\mathcal{I} \models \mathcal{P}$. Since $\mathcal{P}$ is stratified, there exists a unique model $\mathcal{M}(\mathcal{P})$ of $\mathcal{P}$ that is the smallest with respect to set inclusion, which we call the *materialisation* of $\mathcal{P}$. This name is justified by the fact that $\mathcal{M}(\mathcal{P})$ can be constructed by iteratively applying the rules of $\mathcal{P}$ stratum by stratum. Specifically, to apply a rule ρ to an interpretation $\mathcal{I}$, we evaluate the body of ρ as a query over $\mathcal{I}$, and, for each query answer, we instantiate the head of ρ and add the resulting fact to $\mathcal{I}$. Then, we can compute $\mathcal{M}(\mathcal{P})$ bottom-up as follows: after initialising $\mathcal{M}(\mathcal{P})$ by the empty set, we consider the strata of $\mathcal{P}$ one by one so that, for each i in the increasing order, we first apply the rules of the stratum $\mathcal{P}[i]$ to $\mathcal{M}(\mathcal{P})$ as long as possible (i.e., until no new facts can be derived) and then move on to the next stratum $\mathcal{P}[i+1]$. Program $\mathcal{P}$ *entails* a fact γ, written $\mathcal{P} \models \gamma$, if $\gamma \in \mathcal{M}(\mathcal{P})$ holds. Given such $\mathcal{P}$ and γ, checking whether $\mathcal{P} \models \gamma$ holds is a key problem in Datalog and $Datalog_{\mathbb{Z}}$ applications, and it is the main subject of this paper.

If program $\mathcal{P}$ does not use arithmetic functions, then such construction of $\mathcal{M}(\mathcal{P})$ always terminates, and this procedure is used for checking fact entailment in many practical Datalog engines. This, however, no longer holds if $\mathcal{P}$ uses arithmetic.

EXAMPLE 2.1. Let $\mathcal{P}$ be a $Datalog_{\mathbb{Z}}$ program containing a fact $B(0)$ and rule $B(m) \rightarrow B(m+1)$, where predicate B has a single numeric position. Applying $\mathcal{P}$ iteratively derives $B(1), B(2), \ldots$ without stopping. As a result, the materialisation $\mathcal{M}(\mathcal{P})$ contains $B(k)$ for each $k \geq 0$ and is thus infinite. $\lhd$

Despite Example 2.1, the construction of $\mathcal{M}(\mathcal{P})$ is still well defined if we consider the possibly infinite 'limit' of rule application for each $\mathcal{P}[i]$. However, such a 'computation' of $\mathcal{M}(\mathcal{P})$ does not give us an algorithm for checking fact entailment in $Datalog_{\mathbb{Z}}$ with full arithmetic. Moreover, one can exploit the fact that $\mathcal{M}(\mathcal{P})$ can be infinite to show that checking fact entailment is undecidable even for positive programs without multiplication and subtraction that use standard predicates with at most one numeric position [8, 14]. Our goal is thus to identify restrictions that, on the one hand, provide us with languages rich enough to capture interesting data analysis problems and, on the other hand, support decidable or even tractable fact entailment.

3. LIMIT-LINEAR DATALOG$_{\mathbb{Z}}$

In this section, we first introduce *limit* $Datalog_{\mathbb{Z}}$, which can be seen as either a semantic or a syntactic restriction of $Datalog_{\mathbb{Z}}$. To overcome the undecidability of entailment, we then restrict the use of multiplication and thus arrive to *limit-linear* programs. Finally, we present several application examples.

3.1 Limit Programs

As illustrated in Example 2.1, one of the main problems in $Datalog_{\mathbb{Z}}$ is that the materialisation of a program can be infinite. Towards a decidable fragment of $Datalog_{\mathbb{Z}}$, we first introduce *limit* programs. As we shall see, the materialisations of such programs can be represented using finite structures.

DEFINITION 3.1. *A predicate is* object *if it has only object positions, and it is* numeric *if its last*

position is numeric and all its other positions are object; moreover, each numeric predicate is either exact *or* limit*, and each limit predicate is either* min *or* max*. A* limit *(Datalog$_\mathbb{Z}$) program is a program that uses only object and numeric predicates, and where all exact predicates are EDB.*

The notions in Definition 3.1 transfer to atoms and literals in the obvious way; for example, a standard atom is max if its predicate is max.

The intuition behind limit predicates is that they keep only the upper, in case of max, or only the lower, in case of min, bounds on the numeric values for each tuple of object arguments. For example, assume that a program $\mathcal{P}$ consists only of facts $C(a, 5)$ and $C(a, 7)$, where C is a numeric predicate and a is an object. If $\mathcal{P}$ is an ordinary *Datalog*$_\mathbb{Z}$ program, then the materialisation $\mathcal{M}(\mathcal{P})$ coincides with $\mathcal{P}$. If, however, $\mathcal{P}$ is limit and C is max, then the semantics of limit *Datalog*$_\mathbb{Z}$ ensures that every model of $\mathcal{P}$ contains the fact $C(a, k)$ for each $k \leq 7$, and moreover $\mathcal{M}(\mathcal{P})$ consists precisely of these facts. Thus, 7 is the limit value of C on a in $\mathcal{M}(\mathcal{P})$, and we can finitely represent $\mathcal{M}(\mathcal{P})$ as just $C(a, 7)$.

The semantics of a limit program $\mathcal{P}$ is defined model theoretically by considering only *limit-closed* interpretations—that is, interpretations $\mathcal{I}$ that, for each fact $C(\mathbf{a}, \ell) \in \mathcal{I}$ with C a max predicate (in $\mathcal{P}$), contain $C(\mathbf{a}, k)$ for each $k \leq \ell$; and analogously for min predicates. Alternatively, the semantics of limit predicates can be axiomatised in *Datalog*$_\mathbb{Z}$ by extending the program with Rule (3) for each max predicate C and analogously for min predicates.

$$C(\mathbf{x}, m) \wedge (n \leq m) \rightarrow C(\mathbf{x}, n) \tag{3}$$

We introduce a useful syntactic shortcut: for C a limit predicate, $\mathbf{t}$ a tuple of object terms of appropriate size, and s a numeric term, the *least upper bound* (*LUB*) expression $\lceil C(\mathbf{t}, s) \rceil$ abbreviates $C(\mathbf{t}, s) \wedge \mathsf{not}\, C(\mathbf{t}, s + k)$, where $k = 1$ if C is max and $k = -1$ if C is min. Since LUB expressions contain negative literals, $\lceil C(\mathbf{t}, s) \rceil$ can be used in a stratum $\mathcal{P}[i]$ of a limit program $\mathcal{P}$ only if C does not occur in a rule head in $\mathcal{P}[j]$ for each $j \geq i$.

The following example illustrates the intuitions of the definitions we presented thus far.

Example 3.2. Let $\mathcal{P}$ be the program containing Rules (1) and (2) from Section 1, and the facts that describe a directed graph with edge lengths using predicate E. When computing the lengths of the shortest paths from a_s, we need not remember the length of each path from a_s: it suffices to keep just the lengths of the shortest paths found so far. Thus, we can make *dst* a min predicate, which is tantamount to extending $\mathcal{P}$ with the following rule.

$$dst(x, m) \wedge (m \leq n) \rightarrow dst(x, n)$$

As a consequence of this change, a fact $dst(a, \ell)$ follows from $\mathcal{P}$ if and only if the distance from the source node a_s to a is *at most* ℓ; hence, each $dst(a, k)$ with $k \geq \ell$ then follows as well. Note that only *dst* is a limit predicate: predicate E is EDB and so it can be exact, which is in fact necessary to correctly encode the graph structure. Finally, using an LUB expression we can query the exact length of a shortest path: $\mathcal{P}$ entails $\lceil dst(a, \ell) \rceil$ if and only if ℓ is the exact length from a_s to a. ◁

We now discuss the technical challenges of dealing with limit predicates. First, note that any limit-closed interpretation containing a fact over a limit predicate is infinite. In particular, the materialisation of the program from Example 2.1 does not change even if we make B a min predicate. A key insight is that the interpretation of a limit predicate is 'contiguous' for each tuple of object arguments; hence, instead of keeping all of these facts, we can remember only the limit values. Moreover, the limit value may not exist; for example, if we make B a max predicate in Example 2.1, then $\mathcal{M}(\mathcal{P})$ contains $B(k)$ for each integer k. However, we can represent such cases using a special symbol ∞. This motivates the following definition.

Definition 3.3. *A* pseudofact *is either a fact or an expression of the form $C(\mathbf{a}, \infty)$ for C a limit predicate and $\mathbf{a}$ a tuple of objects. A* pseudointerpretation *is a set $\mathcal{J}$ of pseudofacts such that $\ell_1 = \ell_2$ for all limit pseudofacts $C(\mathbf{a}, \ell_1)$ and $C(\mathbf{a}, \ell_2)$ in $\mathcal{J}$.*

We stress an important point of Definition 3.3. Intuitively, pseudofacts represent limit values, so two different pseudofacts for the same predicate and object arguments should not appear in any pseudointerpretation together; for example, a pseudointerpretation for the program in Example 3.2 can contain either $dst(a, 5)$ or $dst(a, 7)$, but never both. This, however, leads us to an important observation: a pseudointerpretation is finite whenever the numbers of predicates and object constants are finite. This property of pseudointerpretations is essential for our decidability results in Section 4.

Moreover, it is easy to see that limit-closed interpretations and pseudointerpretations naturally correspond to each other. For example, to convert a pseudointerpretation $\mathcal{J}$ to a limit-closed interpretation $\mathcal{I}$, we replace each max (pseudo)fact $C(\mathbf{a}, \ell)$ in $\mathcal{J}$ with $\ell \in \mathbb{Z}$ by all facts $C(\mathbf{a}, k)$ with $k \leq \ell$, each such min fact by all $C(\mathbf{a}, k)$ with $k \geq \ell$, and each limit pseudofact $C(\mathbf{a}, \infty)$ by all $C(\mathbf{a}, k)$ with $k \in \mathbb{Z}$.

Conversion in the other direction can be done in a similar way. This allows us to transfer all definitions and notations for limit-closed interpretations to pseudointerpretations; for example, a pseudointerpretation $\mathcal{J}$ *entails* a fact γ if the corresponding limit-closed interpretation $\mathcal{I}$ entails γ, $\mathcal{J}$ is a *pseudomodel* of a limit program $\mathcal{P}$ if $\mathcal{I} \models \mathcal{P}$, and the *pseudomaterialisation* $\mathcal{N}(\mathcal{P})$ is the pseudointerpretation corresponding to the materialisation $\mathcal{M}(\mathcal{P})$.

Finally, we observe that each limit program can be easily rewritten into an equivalent *homogeneous* program that uses only max (or only min) predicates. This can be done by replacing all min predicates with fresh max predicates of the same arities and negating the corresponding numeric arguments in atoms with the replaced predicates.

3.2 Limit-Linear Programs

The ability to finitely represent materialisations does not, however, ensure decidability.

THEOREM 3.4. *The fact entailment problem for positive limit programs is undecidable.*

Theorem 3.4 is due to the fact that limit programs allow multiplication of numeric variables, which we use to reduce the well-known undecidable problem of solving Diophantine equations (i.e., finding integer roots of polynomials) to fact entailment. This motivates the following *linearity* restriction.

DEFINITION 3.5. *A numeric variable m is* guarded *in a rule if it occurs in the rule body in either a function-free positive exact literal or an LUB expression of the form $\lceil C(\mathbf{t}, m)\rceil$. A rule or program is* limit-linear *(*LL-*) if, in each multiplication, at most one argument mentions an unguarded variable.*

Intuitively, a guarded variable m in a rule of an LL-program $\mathcal{P}$ can be matched only to finitely many integers during the evaluation of $\mathcal{P}$. To see this, first note that all exact predicates are EDB in $\mathcal{P}$; thus, if m is guarded because it occurs in a function-free positive literal over an exact predicate, then m can be matched only to facts explicitly mentioned in $\mathcal{P}$. Second, if m is guarded because it occurs in an LUB expression $\lceil C(\mathbf{t}, m)\rceil$, then variable m can be matched to the limit value of C for each valuation of $\mathbf{t}$; moreover, since $\lceil C(\mathbf{t}, m)\rceil$ abbreviates a conjunction containing $\mathsf{not}\, C(\mathbf{t}, m + k)$ for $k = \pm 1$, the limit values for C are fully determined by the strata of $\mathcal{P}$ preceding the stratum of the rule. Note that atoms with other numeric terms involving m, such as $B(\mathbf{t}, m + 1)$, do not make m guarded.

To understand how guarded variables are used to ensure decidability, consider an LL-rule ρ containing a numeric term $m \times n$ with numeric variables m and n. Since ρ is limit-linear, at least one of m and n must be guarded. If m is guarded, then, by the previous paragraph, m can be matched to finitely many integers $k_1, \ldots, k_v$ and hence we can replace ρ with its v instances where the term $m \times n$ is replaced by $k_i \times n$. We have thus reduced multiplication of variables $m \times n$ to *linear* multiplication $k_i \times n$, which allows us to obtain decidability in Section 4 using methods from Presburger arithmetic.

To simplify the discussion in the rest of this paper, we assume that each LL-program is normalised so that each exact atom is function-free. This can be achieved by replacing each positive exact body atom $B(\mathbf{a}, s)$ with s containing functions by the conjunction $B(\mathbf{a}, m) \wedge (m \doteq s)$ with m a fresh numeric variable. Moreover, we note that all exact atoms in rule heads are function-free: the predicates in all these atoms are EDB, and so all such rules are facts.

3.3 Application Examples

Despite the restrictions of Definitions 3.1 and 3.5, LL-programs can still naturally capture many interesting data analysis tasks. We next present five such examples motivated by practical applications.

EXAMPLE 3.6 (DIFFUSION IN NETWORKS). Consider a social network of agents connected by the 'follows' relation. Agent a_s introduces (*tweets*) a message, and each agent a retweets the message if at least k_a agents that a follows tweet this message, where k_a is a positive threshold associated with a. We can determine which agents tweet the message eventually using limit-linear $Datalog_{\mathbb{Z}}$ as follows. We encode the network structure as a dataset $\mathcal{D}_{tw}$ consisting of the object fact $tw(a_s)$ saying that a_s introduces a message, object facts $fol(a, a')$ saying that a follows a', and exact facts $thshld(a, k_a)$ saying that the threshold of a is k_a. We also assume that $\mathcal{D}_{tw}$ is *ordered*—that is, it contains facts $fst(a_1)$, $nxt(a_1, a_2)$, …, $nxt(a_{c-1}, a_c)$, $lst(a_c)$ for some enumeration $a_1, \ldots, a_c$ of all objects (i.e., agents) in $\mathcal{D}_{tw}$. The LL-program $\mathcal{P}_{tw}$, consisting of Rules (4)–(8), encodes message propagation. Here, ac is an 'accumulating' max predicate such that $ac(a, a', m)$ is *true* if there are at least m agents tweeting the message among the agents that a follows and that (inclusively) precede a' in the dataset order.

$$fol(x, y') \wedge fst(y) \rightarrow ac(x, y, 0) \quad (4)$$

$$tw(y) \wedge fol(x, y) \wedge fst(y) \rightarrow ac(x, y, 1) \quad (5)$$

$$ac(x, y', m) \wedge nxt(y', y) \rightarrow ac(x, y, m) \quad (6)$$

$$tw(y) \wedge fol(x, y) \wedge ac(x, y', m) \wedge nxt(y', y) \rightarrow ac(x, y, m + 1) \quad (7)$$

$$thshld(x, m) \wedge ac(x, y, m) \rightarrow tw(x) \quad (8)$$

Then, $\mathcal{P}_{tw} \cup \mathcal{D}_{tw} \models tw(a)$ if and only if an agent a tweets the message according to $\mathcal{D}_{tw}$. $\triangleleft$

Example 3.7 (Bill of materials).
Let $\mathcal{D}_{bm}$ be a dataset describing parts needed to manufacture an end product. Specifically, for each part a and each subpart a' of a, $\mathcal{D}_{bm}$ contains object facts $pt(a)$ and $pt(a')$, and an exact fact $dsp(a, a', k)$ indicating that a needs k copies of a'; also, let $\mathcal{D}_{bm}$ be ordered as in Example 3.6. The graph formed by predicate *dsp* is acyclic and has positive edge weights. Rules (9)–(14) form the LL-program $\mathcal{P}_{bm}$. They compute, using max predicates *ac* and *sp*, how many copies of each subpart are used for each part in total. Intuitively, $ac(a, a', b, k)$ is *true* if part a contains at least k copies of subpart a' in all direct subparts of a that precede part b in the order.

$$pt(x) \to sp(x, x, 1) \tag{9}$$

$$pt(x) \wedge pt(y) \wedge fst(z) \to ac(x, y, z, 0) \tag{10}$$

$$dsp(x, z, n_1) \wedge sp(z, y, n_2) \wedge fst(z) \to ac(x, y, z, n_1 \times n_2) \tag{11}$$

$$ac(x, y, z', m) \wedge nxt(z', z) \to ac(x, y, z, m) \tag{12}$$

$$dsp(x, z, n_1) \wedge sp(z, y, n_2) \wedge ac(x, y, z', m) \wedge nxt(z', z) \to ac(x, y, z, m + n_1 \times n_2) \tag{13}$$

$$ac(x, y, z, m) \to sp(x, y, m) \tag{14}$$

Then, $\mathcal{P}_{bm} \cup \mathcal{D}_{bm} \models sp(a, a', k)$ if and only if a contains at least k copies of a'. Program $\mathcal{P}_{bm}$ is limit-linear since n_1 occurs in positive exact literals over *dsp* and is thus guarded in (11) and (13). $\triangleleft$

Example 3.8 (Counting paths).
Limit-linear $Datalog_{\mathbb{Z}}$ can also count the paths between pairs of nodes in a directed acyclic graph. We encode such a graph in the obvious way as a dataset $\mathcal{D}_{cp}$ that uses a unary object predicate *nd* for nodes and a binary object predicate E for edges; moreover, $\mathcal{D}_{cp}$ is ordered as before. The LL-program $\mathcal{P}_{cp}$, consisting of Rules (15)–(20) with max predicates *pn* and *ac*, counts the paths. Intuitively, $ac(a, a', b, k)$ is *true* if the sum of the numbers of paths from each node b' preceding node b (according to the dataset order) to node a' for which there exists an edge from node a to b' is at least k.

$$nd(x) \to pn(x, x, 1) \tag{15}$$

$$nd(x) \wedge nd(y) \wedge fst(z) \to ac(x, y, z, 0) \tag{16}$$

$$E(x, z) \wedge pn(z, y, n) \wedge fst(z) \to ac(x, y, z, n) \tag{17}$$

$$ac(x, y, z', m) \wedge nxt(z', z) \to ac(x, y, z, m) \tag{18}$$

$$E(x, z) \wedge pn(z, y, n) \wedge ac(x, y, z', m) \wedge nxt(z', z) \to ac(x, y, z, m + n) \tag{19}$$

$$ac(x, y, z, m) \to pn(x, y, m) \tag{20}$$

Then, $\mathcal{P}_{cp} \cup \mathcal{D}_{cp} \models pn(a, a', k)$ if and only if there are at least k paths from node a to node a'. $\triangleleft$

All examples provided thus far use positive LL-programs. In contrast, the following two examples demonstrate the use of stratified negation.

Example 3.9 (Shortest paths).
We modify the program from Section 1 to compute not just the shortest distance, but also the actual paths from a given source node a_s to a given target node a_t in a directed graph with weighted edges. We assume that a dataset $\mathcal{D}_{csp}$ encodes a directed graph with positive edge weights using a ternary exact predicate E as before, and that it identifies the source and target nodes using object facts $src(a_s)$ and $tgt(a_t)$, respectively. The LL-program $\mathcal{P}_{csp}$, consisting of Rule (2) and Rules (21)–(23), computes a directed acyclic graph G with source a_s and target a_t, encoded using a binary object predicate *spE*, such that every maximal path in G is a shortest path from a_s to a_t in the original graph.

$$src(x) \to dst(x, 0) \tag{21}$$

$$\lceil dst(x, m) \rceil \wedge E(x, y, n) \wedge \lceil dst(y, m + n) \rceil \wedge tgt(y) \to spE(x, y) \tag{22}$$

$$\lceil dst(x, m) \rceil \wedge E(x, y, n) \wedge \lceil dst(y, m + n) \rceil \wedge spE(y, z) \to spE(x, y) \tag{23}$$

The first stratum consists of Rules (2) and (21), and computes the length of a shortest path from a_s to each other node using the min predicate *dst*; in particular, we have that $\mathcal{P}_{csp} \cup \mathcal{D}_{csp} \models \lceil dst(a, k) \rceil$ if and only if k is the length of a shortest path from a_s to a. Then, the second stratum, consisting of Rules (22) and (23), computes predicate *spE* such that $\mathcal{P}_{csp} \cup \mathcal{D}_{csp} \models spE(a, a')$ if the edge (a, a') is a part of a shortest path from a_s to a_t. $\triangleleft$

Example 3.10 (Closeness centrality).
The closeness centrality of a node in a strongly connected directed graph G with weighted edges is a measure of how central the node is in the graph [23]. Variants of this measure are useful, for example, for the analysis of market potential. The *closeness centrality* of a node a is $1 \,/\, \sum_{a' \text{ node in } G} \Omega(a, a')$, where $\Omega(a, a')$ is the length of a shortest path from a to a'; the sum in the denominator is often called the *farness centrality* of a. We next present an LL-program $\mathcal{P}_{cc}$ that identifies a node of maximal closeness centrality in a strongly connected directed graph with weighted edges. We encode such a graph as an ordered dataset $\mathcal{D}_{cc}$ using a unary object predicate *nd* and a ternary exact predicate E. Program

$\mathcal{P}_{cc}$ consists of Rules (24)–(32), where *dst*, *fns* and *fns′* are min, and *cntr* and *cntr′* are object.

$$nd(x) \rightarrow dst(x, x, 0) \tag{24}$$
$$dst(x, y, m) \wedge E(y, z, n) \rightarrow dst(x, z, m{+}n) \tag{25}$$
$$nd(x) \wedge fst(y) \wedge dst(x, y, n) \rightarrow fns'(x, y, n) \tag{26}$$
$$fns'(x, y, m) \wedge nxt(y, z) \wedge dst(x, z, n) \rightarrow fns'(x, z, m{+}n) \tag{27}$$
$$fns'(x, y, n) \wedge lst(y) \rightarrow fns(x, n) \tag{28}$$
$$fst(x) \rightarrow cntr'(x, x) \tag{29}$$
$$cntr'(x, z) \wedge nxt(x, y) \wedge \lceil fns(z, m) \rceil \wedge \lceil fns(y, n) \rceil \wedge (n < m) \rightarrow cntr'(y, y) \tag{30}$$
$$cntr'(x, z) \wedge nxt(x, y) \wedge \lceil fns(z, m) \rceil \wedge \lceil fns(y, n) \rceil \wedge (m \leq n) \rightarrow cntr'(y, z) \tag{31}$$
$$cntr'(x, z) \wedge lst(x) \rightarrow cntr(z) \tag{32}$$

The first stratum consists of Rules (24)–(28); Rules (24) and (25) compute the distance between any two nodes, and Rules (26)–(28) then compute the farness centrality of each node based on the aforementioned distances. In the second stratum, (29)–(32), $\mathcal{P}_{cc}$ uses negation (hidden inside the LUB expressions) to compute a node of minimal farness centrality (and hence of maximal closeness centrality), which is recorded using the *cntr* predicate. ◁

4. COMPLEXITY OF LL-PROGRAMS

We now discuss the complexity of fact entailment in limit-linear $Datalog_{\mathbb{Z}}$. We consider *combined complexity*, where the input consists of an LL-program $\mathcal{P}$ and a fact γ, and *data complexity*, where we assume that $\mathcal{P} = \mathcal{P}' \cup \mathcal{D}$ for a dataset $\mathcal{D}$ and a fixed LL-program $\mathcal{P}'$ (i.e., only $\mathcal{D}$ and γ are given as input). We assume binary coding of integers and write $\|\mathcal{P}\|$ for the size of the representation of a program $\mathcal{P}$; however, our results also hold for unary coding. We show that, for the full language, the problem is complete for $\Delta_2^{\mathsf{EXP}} = \mathsf{EXP}^{\mathsf{NP}}$ in combined and for $\Delta_2^{\mathsf{P}} = \mathsf{P}^{\mathsf{NP}}$ in data complexity, while, for the positive and semi-positive fragments, it is complete for coNEXP and for coNP, respectively. Thus, (semi-)positive LL-programs have the same complexity as answer-set programming [24], and are one level above the usual (semi-)positive Datalog, which is EXP- and P-complete [8]. Moreover, in terms of complexity, LL-programs are between the usual and disjunctive answer-set programming, where the latter is complete for Π_2^{EXP} and Π_2^{P} [10].

4.1 Positive Fragment

We first consider the problem of checking entailment $\mathcal{P} \models \gamma$ of a fact γ by a positive LL-program $\mathcal{P}$. Reasoning algorithms for usual Datalog often start with computing the program *grounding*—that is, replacing all variables with constants in the program in all possible ways. This variable elimination simplifies rule application, but with a cost of an exponential blow-up (in the size of the program). Our algorithms use a similar preprocessing step. However, grounding in a usual way would require replacing each numeric variable in an LL-program with every integer, which would result in an infinite grounding. To avoid working with infinite programs, we need to adapt the notion of grounding.

Definition 4.1. *An LL-rule or LL-program is* object-and-guarded-ground *(*OG-ground*) if it has neither object nor guarded numeric variables. The* canonical OG-grounding *of an LL-program $\mathcal{P}$ is the OG-ground program $\mathcal{G}(\mathcal{P})$ that contains the OG-ground instance $\rho\sigma$ for each rule $\rho \in \mathcal{P}$ and each substitution σ mapping all object and guarded numeric variables of ρ to constants in $\mathcal{P}$.*

To produce $\mathcal{G}(\mathcal{P})$ for a positive LL-program $\mathcal{P}$, we replace, in all possible ways, all object variables in $\mathcal{P}$ with objects in $\mathcal{P}$ and all guarded numeric variables with integers in $\mathcal{P}$. The discussion in Section 3.2 explains why considering only the integers from $\mathcal{P}$ suffices. It is easy to see that $\mathcal{P}$ and $\mathcal{G}(\mathcal{P})$ are equivalent in the sense that $\mathcal{M}(\mathcal{P}) = \mathcal{M}(\mathcal{G}(\mathcal{P}))$, so $\mathcal{P} \models \gamma$ if and only if $\mathcal{G}(\mathcal{P}) \models \gamma$ for each fact γ.

Now we show how to apply a positive OG-ground rule ρ to a pseudointerpretation $\mathcal{J}$. A minor problem is that, if we derive a fact $C(\mathbf{a}, \ell)$ with C max and we already have a fact $C(\mathbf{a}, k)$ with $k < \ell$, then we must remove $C(\mathbf{a}, k)$, and similarly if C is min. More importantly, identifying the substitutions that match the body of ρ to $\mathcal{J}$ is considerably more complex than for ordinary interpretations. For example, the body of the rule $C_1(m) \wedge C_2(m) \rightarrow C(m)$ where C_1 and C_2 are max predicates does not match to the pseudointerpretation $\{C_1(7), C_2(5)\}$ directly, but the rule is applicable and it derives $C(5)$. We address this difficulty by reducing the problem of a positive OG-ground rule application to *integer linear optimisation*. Specifically, to match ρ to $\mathcal{J}$, we can transform ρ into a system of integer linear inequalities $\psi(\rho, \mathcal{J})$ whose solutions encode exactly the substitutions obtained by matching of the body of ρ to (the limit-closed interpretation corresponding to) $\mathcal{J}$. Thus, to compute the consequences of ρ with a limit atom in the head, we just need the solution that optimises the numeric term in this atom.

Unfortunately, as shown in Example 2.1, iterative rule application may not terminate even for OG-ground programs. So, while one can formally

describe a process that constructs the (finite) pseudomaterialisation by iteratively applying rules, this process may be infinite and thus does not provide us with a decision procedure for fact entailment.

We next discuss a key insight about positive LL-programs, which we use to bound the magnitudes of integers in pseudomaterialisations. We exploit a connection with *Presburger arithmetic*—a theory of first-order formulas with numeric variables (i.e., all variables range over integers) where all atoms are comparisons of the form $(s_1 \leq s_2)$ or $(s_1 < s_2)$ (i.e., as in $Datalog_{\mathbb{Z}}$) with multiplication-free numeric terms s_1 and s_2. Due to the lack of multiplication, reasoning in this theory is decidable.

Now, to check entailment $\mathcal{P} \models \gamma$ for a positive LL-program $\mathcal{P}$ and a fact γ, we encode $\mathcal{P}$ as a Presburger formula $\xi_{\mathcal{P}}$ so that each pseudomodel of $\mathcal{P}$ corresponds to a valuation of the free variables of $\xi_{\mathcal{P}}$ that makes $\xi_{\mathcal{P}}$ true. We analogously encode γ as a formula ξ_γ, and we thus reduce $\mathcal{P} \models \gamma$ to checking whether the Presburger sentence $\forall \mathbf{v}. (\xi_{\mathcal{P}} \rightarrow \xi_\gamma)$, where $\mathbf{v}$ are the free variables of $\xi_{\mathcal{P}}$ and ξ_γ, is true. To illustrate a simplified version of our encoding, consider an LL-program $\mathcal{P}$ consisting of a fact $C(2)$ and a rule $C(m) \rightarrow D(m+2)$, and a fact $\gamma = D(3)$, where both C and D are max. We encode the values of C and D in a pseudomodel of $\mathcal{P}$ using variables v_C and v_D, respectively. Fact $C(2)$ says 'the value of C in each pseudomodel is at least 2', so we encode it as $\xi_1 = (2 \leq v_C)$. We also encode the rule as $\xi_2 = \forall m. (m \leq v_C) \rightarrow (m + 2 \leq v_D)$, and γ as $\xi_\gamma = (3 \leq v_D)$. Clearly, $\mathcal{P} \models \gamma$ if and only if $\forall v_C \forall v_D. (\xi_{\mathcal{P}} \rightarrow \xi_\gamma)$ is true for $\xi_{\mathcal{P}} = \xi_1 \wedge \xi_2$. Following this idea, our encoding uses additional variables to represent undefined and unbounded values.

Thus, entailment $\mathcal{P} \models \gamma$ for a positive LL-program $\mathcal{P}$ and fact γ is decidable since Presburger arithmetic is decidable; however, the complexity bounds derived from the standard reasoning algorithms for Presburger arithmetic are not optimal. Instead, by analysing the structure of our encoding and applying the results by Chistikov and Haase [5], we show that $\forall \mathbf{v}. (\xi_{\mathcal{P}} \rightarrow \xi_\gamma)$ is true if and only if it is true when the sentence is evaluated over integers with magnitudes at most double exponential in $\|\mathcal{P}\| + \|\gamma\|$, and at most exponential in $\|\mathcal{D}\| + \|\gamma\|$, where $\mathcal{D}$ is the dataset part of $\mathcal{P}$.[1] So, each integer can be written in binary using number of bits exponential in $\|\mathcal{P}\| + \|\gamma\|$ and polynomial in $\|\mathcal{D}\| + \|\gamma\|$.

Since the valuations of the free variables of $\xi_{\mathcal{P}}$ encode the pseudomodels of $\mathcal{P}$, nonentailment $\mathcal{P} \not\models \gamma$ is witnessed by a pseudomodel $\mathcal{J}$ of $\mathcal{P}$ where the integers are bounded in the same way (note that ∞ is not an integer so $\mathcal{J}$ can contain ∞). Thus, we can decide $\mathcal{P} \not\models \gamma$ by first guessing a pseudointerpretation $\mathcal{J}$ over the signature of $\mathcal{P}$ with integers bounded as explained, and then checking that $\mathcal{J} \models \mathcal{G}(\mathcal{P})$ and $\mathcal{J} \not\models \gamma$. Overall, $\|\mathcal{J}\|$ is at most exponential in $\|\mathcal{P}\| + \|\gamma\|$ and at most polynomial in $\|\mathcal{D}\| + \|\gamma\|$. Moreover, to check $\mathcal{J} \models \mathcal{G}(\mathcal{P})$, we apply the rules of $\mathcal{G}(\mathcal{P})$ to $\mathcal{J}$ and verify that no new fact is derived, which requires solving integer optimisation problems. Thus, our algorithm works in NEXP in combined and in NP in data complexity.

For the matching lower data complexity bound, we reduce the complement of the canonical NP-complete problem SAT. Given an arbitrary propositional formula Φ with h variables, we (arbitrarily) order all variables of Φ so we can view each variable assignment σ as an integer $\ell(\sigma)$ between 0 and $2^h - 1$. To decide the satisfiability of Φ, we then can enumerate all assignments σ in the increasing order of $\ell(\sigma)$ until we either find a σ satisfying Φ, or we find that σ_{max} with $\ell(\sigma_{max}) = 2^h - 1$ does not satisfy Φ. If Φ is encoded in a dataset, this enumeration can be simulated by a fixed positive LL-program that stores $\ell(\sigma)$ in the numeric argument of a max predicate, starting with 0 and incrementing if and only if the current σ does not satisfy Φ. Hence, Φ is unsatisfiable if and only if our encoding entails a fact with a numeric argument 2^h.

As required for data complexity, the program in this reduction does not depend on input Φ. In contrast, this is not necessary for combined complexity, and the same ideas can be applied to reduce the succinct version of SAT—a canonical NEXP-complete problem [21]. We arrive to the following result.

THEOREM 4.2. *The fact entailment problem for positive LL-programs is* coNEXP*-complete in combined and* coNP*-complete in data complexity.*

4.2 Semi-Positive and Full Fragments

We first consider semi-positive programs, where negation can be used only over EDB predicates. We reduce our problem to the positive case by computing the canonical OG-grounding of the given program and 'evaluating' all negative literals. This idea is captured by the following definition.

DEFINITION 4.3. *The* reduct $\mathcal{R}(\mathcal{P})$ *of a semi-positive LL-program $\mathcal{P}$ is the positive OG-program obtained from $\mathcal{G}(\mathcal{P})$ by applying the following to each rule $\rho \in \mathcal{G}(\mathcal{P})$ and each negative literal* $\mathsf{not}\,\alpha$ *in ρ.*

- *Let α be an object atom. If $\alpha \notin \mathcal{G}(\mathcal{P})$, then delete* $\mathsf{not}\,\alpha$ *from ρ, and otherwise delete ρ.*
- *Let $\alpha = B(\mathbf{a}, s)$ be an exact atom and consider all integers $k_1 < \cdots < k_h$ such that $B(\mathbf{a}, k_i) \in \mathcal{G}(\mathcal{P})$*

[1] We thank Christoph Haase for providing the proof of a key lemma for this statement.

holds for each $i \in [1, h]$. *If* $h = 0$, *then delete* $\mathsf{not}\,\alpha$ *from* ρ; *otherwise, replace* ρ *by* $h + 1$ *rules obtained by replacing* $\mathsf{not}\,\alpha$ *in* ρ *with* $(s < k_1)$, $(k_{i-1} < s < k_i)$ *for* $i \in [2, h]$, *and* $(k_h < s)$.

– *Let* $\alpha = C(\mathbf{a}, s)$ *be a limit atom and consider the set* $S = \{\ell \in \mathbb{Z} \mid C(\mathbf{a}, \ell) \in \mathcal{G}(\mathcal{P})\}$ *of integers. If* $S = \emptyset$, *then delete* $\mathsf{not}\,\alpha$ *from* ρ; *otherwise, replace* $\mathsf{not}\,\alpha$ *in* ρ *with atom* $(\max S < s)$ *if* C *is* max, *and with atom* $(\min S > s)$ *if* C *is* min.

By construction, $\mathcal{M}(\mathcal{P}) = \mathcal{M}(\mathcal{R}(\mathcal{P}))$ for each semi-positive LL-program $\mathcal{P}$, so $\mathcal{P} \models \gamma$ is equivalent to $\mathcal{R}(\mathcal{P}) \models \gamma$ for each fact γ. Moreover, $\mathcal{R}(\mathcal{P})$ is positive and OG-ground, and can be computed with the same bounds as $\mathcal{G}(\mathcal{P})$. Thus, the upper bounds of Theorem 4.2 transfer to semi-positive programs.

THEOREM 4.4. *The fact entailment problem for semi-positive LL-programs is in* coNEXP *in combined and in* coNP *in data complexity.*

To handle arbitrary LL-programs, we first show that the integer bound from Section 4.1 applies not only to some pseudomodel, but also to the pseudo-materialisation of each positive LL-program. Then, given a fact γ and an arbitrary LL-program $\mathcal{P}$ with strata $\mathcal{P}[1], \dots, \mathcal{P}[h]$, we decide $\mathcal{P} \models \gamma$ as follows. For each $i \in [1, h]$, first, we let $\mathcal{P}'_i = \mathcal{R}(\mathcal{P}[i] \cup \mathcal{J}_{i-1})$ (assuming $\mathcal{J}_0 = \emptyset$); then, we add to the pseudointerpretation $\mathcal{J}_i$ each max pseudofact $C(\mathbf{a}, \ell)$ such that $\mathcal{P}'_i \models C(\mathbf{a}, \ell)$ and ℓ is either ∞ or an integer bounded as above satisfying $\mathcal{P}'_i \not\models C(\mathbf{a}, \ell + 1)$; finally, we add to $\mathcal{J}_i$ pseudofacts of other types analogously. The pseudofacts in $\mathcal{P}[i] \cup \mathcal{J}_{i-1}$ can contain symbol ∞, but Definition 4.3 generalises to such 'programs' without change. Clearly, $\mathcal{J}_h = \mathcal{N}(\mathcal{P})$, and so $\mathcal{P} \models \gamma$ if and only if $\mathcal{J}_h \models \gamma$. A naïve complexity bound of this algorithm is nonelementary, but a fine-grained analysis (in particular, bounding $\|\mathcal{J}_i\|$) gives the upper bounds of Theorem 4.5.

For the lower bounds, we generalise the ideas of the positive case. In particular, for data complexity, we reduce the canonical Δ_2^{P}-complete problem ODD-MAX-SAT, where the question is if the maximum value of $\ell(\sigma)$ over all assignments σ satisfying a propositional formula Φ is odd. For the combined complexity, we use a similar reduction of the Δ_2^{EXP}-complete succinct version of ODD-MAX-SAT.

THEOREM 4.5. *The fact entailment problem for LL-programs is* Δ_2^{EXP}*-complete in combined and* Δ_2^{P}*-complete in data complexity.*

5. TRACTABLE FRAGMENTS

Tractability of reasoning in data complexity is important for problems involving large datasets. We now present a *stability* condition on LL-programs that brings the complexity of reasoning down to EXP in combined and to P in data complexity, thus matching the bounds for usual Datalog. We then present a syntactic *type-consistency* condition that ensures stability and is simple to check.

5.1 Stable LL-Programs

The fact entailment algorithm for usual Datalog computes the materialisation iteratively, which can be done in polynomial time in the size of data. We now present a further restriction on LL-programs that makes such iterative computation viable for LL-programs. The key difficulty in doing so is to detect when a numeric argument *diverge*—that is, when it increases or decreases indefinitely. Hence, to ensure tractability, we must be able to detect divergence after polynomially many rule applications. Example 5.1 illustrates the problem of divergence.

EXAMPLE 5.1. Consider an LL-program $\mathcal{P}_{st}$ with the following rules with max predicates C_1 and C_2.

$$C_1(0) \quad C_1(m) \to C_2(m) \quad C_2(m) \to C_1(m+1)$$

Both C_1 and C_2 diverge when computing pseudo-materialisation $\mathcal{N}(\mathcal{P}_{st})$ due to a cyclic dependency between C_1 and C_2. The existence of such a dependency, however, may not lead to divergence. Let an LL-program $\mathcal{P}'_{st}$ be obtained from $\mathcal{P}_{st}$ by adding a fact $C(5)$, for C max, and replacing the second rule by the following rule.

$$C_1(m) \wedge C(m) \to C_2(m)$$

While C_1 and C_2 still depend on each other, the increase in C_1 and C_2 is bounded by an independent value of C, so neither C_1 nor C_2 diverges. ◁

To capture these ideas formally, we first extend $\mathbb{Z} \cup \{\infty\}$ with a new symbol $\bot$, which indicates that a fact does not hold for any integer. We also define $\bot < k < \infty$ for each $k \in \mathbb{Z}$; $\bot + \ell = \bot$ and $\infty + \ell = \infty$ for each $\ell \in \mathbb{Z} \cup \{\infty\}$; and $\bot + \infty = \bot$.

Next we formalise the notion of dependency: a numeric variable m *depends* on a numeric variable n in an OG-ground rule ρ if $m = n$ or m occurs in an atom in ρ with a variable that depends on n. A numeric term s_2 *depends* on a numeric term s_1 if s_2 mentions a variable depending on a variable in s_1.

We next introduce a key notion of a value propagation graph. Our definition is based on the system of linear inequalitites $\psi(\rho, \mathcal{J})$ from Section 4.1 whose solutions encode matches of the body of an OG-ground rule ρ to a pseudointerpretation $\mathcal{J}$. So, if the head of ρ is a limit atom $C(\mathbf{a}, s)$, then $\psi(\rho, \mathcal{J})$

corresponds to an integer linear optimisation problem $\psi^*(\rho, \mathcal{J})$ that optimises (i.e., maximises or minimises, depending on the type of C) the value of s under $\psi(\rho, \mathcal{J})$. If $\psi(\rho, \mathcal{J})$ is satisfiable (i.e., ρ is applicable to $\mathcal{J}$), then $\psi^*(\rho, \mathcal{J})$ can either be bounded and have an optimal integer value, or unbounded.

Definition 5.2. *Given an OG-ground program $\mathcal{P}$ and a pseudointerpretation $\mathcal{J}$, the* value propagation graph *of $\mathcal{P}$ over $\mathcal{J}$ is the weighted directed graph (V, E, Ω) with the following components.*

- *The set of nodes V contains a node $\langle C\mathbf{a}\rangle$ for each limit atom $C(\mathbf{a}, s)$ in a rule head in $\mathcal{P}$.*
- *The set of edges E contains $(\langle C_1\mathbf{a}_1\rangle, \langle C_2\mathbf{a}_2\rangle)$ for each rule ρ in $\mathcal{P}$ with satisfiable $\psi(\rho, \mathcal{J})$ that* produces *the edge—that is, has $C_1(\mathbf{a}_1, s_1)$ in the body and $C_2(\mathbf{a}_2, s_2)$ in the head where s_2 depends on s_1.*
- *The weight $\Omega(e)$ of each edge $e = (\langle C_1\mathbf{a}_1\rangle, \langle C_2\mathbf{a}_2\rangle)$ in E is an element of $\mathbb{Z} \cup \{\bot, \infty\}$ defined as*

$$\Omega(e) = \max\{\Omega_\rho(e) \mid \rho \in \mathcal{P} \text{ produces } e\},$$

where $\Omega_\rho(e)$ is defined as follows, for $\ell \in \mathbb{Z} \cup \{\infty\}$ with $C_1(\mathbf{a}_1, \ell) \in \mathcal{J}$ (which exists by applicability):
  - *$\Omega_\rho(e) = \infty$ if $\psi^*(\rho, \mathcal{J})$ is unbounded,*
  - *$\Omega_\rho(e) = \bot$ if $\psi^*(\rho, \mathcal{J})$ is bounded and $\ell = \infty$,*
  - *$\Omega_\rho(e) = (-1)^{d_2} \cdot k - (-1)^{d_1} \cdot \ell$ if $\psi^*(\rho, \mathcal{J})$ has optimal value k and $\ell \in \mathbb{Z}$ where, for $i \in \{1, 2\}$, d_i is 0 if C_i is* max *and 1 if C_i is* min.

The weight $\Omega(\Pi)$ *of a path Π in a value propagation graph is the sum of the edge weights along Π; Π has* positive weight *if $\Omega(\Pi)$ is a positive integer or ∞.*

Intuitively, graph (V, E, Ω) of a OG-ground program $\mathcal{P}$ over a pseudointerpretation $\mathcal{J}$ describes how, for each pseudofact $C_1(\mathbf{a}_1, \ell)$ in $\mathcal{J}$, applying $\mathcal{P}$ propagates ℓ to other pseudofacts. For example, every edge $e = (\langle C_1\mathbf{a}_1\rangle, \langle C_2\mathbf{a}_2\rangle)$ with max C_1 and C_2 indicates that a rule is applicable to a fact $C_1(\mathbf{a}_1, \ell) \in \mathcal{J}$ and that it produces $C_2(\mathbf{a}_2, \ell + \Omega(e))$.

It is easy to check that the value propagation graph increases monotonically during rule application in the following sense: for each OG-ground program $\mathcal{P}$ and pseudointerpretations $\mathcal{J}$ and $\mathcal{J}'$ such that $\mathcal{I} \subseteq \mathcal{I}'$ for the corresponding limit-closed interpretations $\mathcal{I}$ and $\mathcal{I}'$, we always have $V = V'$ and $E \subseteq E'$ for the graphs (V, E, Ω) and (V', E', Ω') of $\mathcal{P}$ over $\mathcal{J}$ and $\mathcal{P}$ over $\mathcal{J}'$, respectively. This guarantees the correctness of the following definition.

Definition 5.3. *An OG-ground program $\mathcal{P}$ is* stable *if, for all pseudointerpretations $\mathcal{J}$ and $\mathcal{J}'$ such that $\mathcal{I} \subseteq \mathcal{I}'$ holds for the corresponding limit-closed interpretations $\mathcal{I}$ and $\mathcal{I}'$, for the value propagation graphs (V, E, Ω) and (V, E', Ω') of $\mathcal{P}$ over $\mathcal{J}$ and of $\mathcal{P}$ over $\mathcal{J}'$, respectively, and for each edge $e \in E$, it is the case that $\Omega(e) \leq \Omega'(e)$ holds.*

Intuitively, iterative rule applications never decrease the edge weights if a program is stable. Program $\mathcal{P}_{st}$ in Example 5.1 is stable, while $\mathcal{P}'_{st}$ is not stable since $\Omega(e) = 0$ and $\Omega'(e) = -1$ for the edge $e = (\langle C_1\rangle, \langle C_2\rangle)$ in the propagation graphs (V, E, Ω) and (V, E', Ω') of $\mathcal{P}'_{st}$ over the pseudointerpretations $\{C_1(0), C(0)\}$ and $\{C_1(1), C(0)\}$, respectively.

It can be shown that a positive-weight cycle of a stable OG-ground program $\mathcal{P}$ over a pseudointerpretation $\mathcal{J}$ guarantees divergence of numeric arguments along the cycle by repeated application of the rules of $\mathcal{P}$ to $\mathcal{J}$. This allows us to compute the pseudomaterialisation in a finite number of steps by applying the rules iteratively, provided that, after each rule application, we construct the value propagation graph for the current pseudointerpretation and bump to ∞ the numeric arguments of all pseudofacts along each positive-weight cycle. It follows that the number of rule applications needed to obtain such a cycle can be polynomially bounded in the size of $\mathcal{P}$, which leads to the following lemma.

Lemma 5.4. *For each stable OG-ground program $\mathcal{P}$, the pseudomaterialisation $\mathcal{N}(\mathcal{P})$ can be computed in time exponential in $\|\mathcal{P}\|$ and polynomial in $\|\mathcal{D}\|$, where $\mathcal{D}$ is the dataset component of $\mathcal{P}$.*

Using Lemma 5.4, we can decide fact entailment for stable OG-ground programs in EXP in combined and in P in data complexity—that is, with the same complexity as for usual Datalog. We next show how to extend this result to LL-programs with negation. We first generalise the notion of stability.

Definition 5.5. *An LL-program $\mathcal{P}$ is* stable *if it can stratified as $\mathcal{P}[1], \ldots, \mathcal{P}[h]$ such that, for each $i \in [1, h]$ and for the pseudomaterialisation $\mathcal{J}_{i-1}$ of $\mathcal{P}[1] \cup \cdots \cup \mathcal{P}[i-1]$, the reduct $\mathcal{R}(\mathcal{P}[i] \cup \mathcal{J}_{i-1})$ is stable (assuming $\mathcal{J}_0 = \emptyset$ for uniformity).*

Combining Lemma 5.4 with the ideas in Theorem 4.5, we obtain the following result.

Theorem 5.6. *The fact entailment problem for stable LL-programs is* EXP*-complete in combined and* P*-complete in data complexity.*

5.2 Type-Consistent Programs

Stability identifies a large class of LL-programs for which reasoning is tractable. Unfortunately, the condition is semantic, rather than syntactic. Moreover, it is not a local condition in the sense that it cannot be verified by looking at each rule in isolation but depends on how different rules interact. Finally, checking stability of an LL-program involves computing the reduct of each stratum, which depends on the materialisation of the preceding strata

(in fact, even checking stability of an OG-ground program is coNP-hard). This motivates the following sufficient condition for stability.

DEFINITION 5.7. *An* mm-typing *of variables of an LL-rule partitions all unguarded numeric variables occurring in positive limit body literals of the rule into* max *and* min *types. Given such a typing, a numeric term is of type* max *if it is of the form*

$$s + \left(\sum_{i=1}^{v} k_i \times m_i\right) - \left(\sum_{j=1}^{w} \ell_j \times n_j\right),$$

for s a numeric term not mentioning any max *or* min *variables, nonnegative integers v and w, each m_i a* max *variable with coefficient $k_i \geq 1$, and each n_j a* min *variable with coefficient $\ell_i \geq 1$. Moreover, a numeric term is of type* min *if the same holds except that each m_i is* min *and each is n_j* max.

An LL-rule $\rho = \varphi \to \alpha$ is type-consistent *if it has a variable mm-typing with the following properties.*

- *Each numeric variable in each negative exact literal in φ is guarded.*
- *The numeric term of each* max *and each* min *atom in ρ is of type* max *and* min*, respectively.*
- *Each comparison in φ has the form $(s_1 < s_2)$ or $(s_1 \leq s_2)$, for s_1 of type* min *and s_2 of type* max.
- *If $\alpha = C_2(\mathbf{a}_2, s_2)$ is a limit atom then, for each positive limit literal $C_1(\mathbf{a}_1, s_1)$ in φ with s_2 depending on s_1, terms s_1 and s_2 have a common unguarded variable that has coefficient 1 in s_1 and does not occur in any other positive limit literals in φ, where* dependency *is defined as in Section 5.1 except that only unguarded numeric variables are taken into account.*

An LL-program is type-consistent *if all of its rules are type-consistent.*

Type-consistency can be checked rule by rule in L, and all programs in Section 3.3 are type-consistent. Furthermore, the complexity results in Theorem 5.6 apply since type-consistency implies stability.

THEOREM 5.8. *Each type-consistent LL-program is stable.*

6. RELATED WORK

The closest formalism to limit $Datalog_{\mathbb{Z}}$ is the 'monotonic programs' of Ross and Sagiv [22]. Their core fragment extends usual Datalog by predicates whose last position ranges over partially ordered *cost domains* (e.g., integers ordered by $\leq$) with associated built-in functions. They allow only for interpretations resembling our pseudointerpretations, and their programs are required to be *monotonic* in the sense that rule applications should produce only interpretations of the appropriate form and preserve the orders of all cost domains. Unfortunately, checking monotonicity is undecidable, and moreover monotonicity does not imply decidability of fact entailment. Nonetheless, there is a rich common fragment of monotonic programs and limit-linear $Datalog_{\mathbb{Z}}$ that inherits the decidability, complexity, and tractability of the latter.

Another related formalism is Datalog^{FS} proposed by Mazuran et al. [19], which extends usual Datalog with so-called *frequency support goals* and provides the formal underpinning for the DeALS system [31]. Similar to the language of Ross and Sagiv, fact entailment in Datalog^{FS} is undecidable, and no practical decidable fragment has been identified. Finding such fragments could potentially be accomplished by transferring some ideas from our work.

$Datalog_{\mathbb{Z}}$ is also closely related to constraint logic programming (CLP). Although a number of decidable CLP languages have been identified [7], none of them allow for recursive *numeric value invention*, which is an integral feature of $Datalog_{\mathbb{Z}}$ necessary to capture several examples in Section 3.3.

Finally, note that some examples from Section 3.3 implement aggregation over recursive rules. Several attempts were made to provide a generic semantics for such aggregation (including monotonic programs and Datalog^{FS} considered above in their full power). However, all these attempts yield solutions either for restricted classes of programs that are subject to strong monotonicity assumptions, use only *min* and *max* aggregate functions, or have undecidable fact entailment [6, 11, 12, 16, 20, 27].

7. CONCLUSION

We have presented several decidable fragments of $Datalog_{\mathbb{Z}}$ that can capture interesting data analysis problems. For future work, we first aim to systematically explore the ability of $Datalog_{\mathbb{Z}}$ to express aggregate functions, which are a necessary component of any practical data analytics formalism. Second, we plan to extend our results to the context of descriptive complexity [13]: we believe that the languages of semi-positive and arbitrary LL-programs capture coNP and Δ_2^{P}, respectively. Another avenue for future work is exploring practical applicability of our algorithms and their implementation in practical declarative data analysis systems.

Acknowledgements This research is supported by EPSRC projects MaSI^3, AnaLOG, ED^3 and OASIS.

8. REFERENCES

[1] P. Alvaro, T. Condie, N. Conway, K. Elmeleegy, J. M. Hellerstein, and R. Sears. BOOM analytics: Exploring data-centric,

declarative programming for the cloud. In *EuroSys*, pages 223–236, 2010.
[2] M. Aref, B. ten Cate, T. J. Green, B. Kimelfeld, D. Olteanu, E. Pasalic, T. L. Veldhuizen, and G. Washburn. Design and implementation of the LogicBlox system. In *SIGMOD*, pages 1371–1382, 2015.
[3] C. Beeri, S. A. Naqvi, O. Shmueli, and S. Tsur. Set constructors in a logic database language. *J. Log. Pr.*, 10(3&4):181–232, 1991.
[4] B. Chin, D. von Dincklage, V. Ercegovac, P. Hawkins, M. S. Miller, F. J. Och, C. Olston, and F. Pereira. Yedalog: Exploring knowledge at scale. In *SNAPL*, pages 63–78, 2015.
[5] D. Chistikov and C. Haase. The taming of the semi-linear set. In *ICALP*, volume 55, pages 128:1–128:13, 2016.
[6] M. P. Consens and A. O. Mendelzon. Low complexity aggregation in GraphLog and Datalog. *Th. Comp. Sci.*, 116(1):95–116, 1993.
[7] J. Cox, K. McAloon, and C. Tretkoff. Computational complexity and constraint logic programming languages. *Ann. Math. Artif. Intell.*, 5(2–4):163–189, 1992.
[8] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.
[9] J. Eisner and N. W. Filardo. Dyna: Extending datalog for modern AI. In *Datalog*, pages 181–220, 2011.
[10] T. Eiter, G. Gottlob, and H. Mannila. Disjunctive datalog. *ACM Trans. Database Syst.*, 22(3):364–418, 1997.
[11] W. Faber, G. Pfeifer, and N. Leone. Semantics and complexity of recursive aggregates in answer set programming. *Artif. Intell.*, 175(1):278–298, 2011.
[12] S. Ganguly, S. Greco, and C. Zaniolo. Extrema predicates in deductive databases. *J. Comput. System Sci.*, 51(2):244–259, 1995.
[13] N. Immerman. *Descriptive Complexity*. Springer, 1999.
[14] M. Kaminski, B. Cuenca Grau, E. V. Kostylev, B. Motik, and I. Horrocks. Foundations of declarative data analysis using limit datalog programs. In *IJCAI*, pages 1123–1130, 2017.
[15] M. Kaminski, B. Cuenca Grau, E. V. Kostylev, B. Motik, and I. Horrocks. Stratified negation in limit Datalog programs. In *IJCAI*, pages 1875–1881, 2018.
[16] D. B. Kemp and P. J. Stuckey. Semantics of logic programs with aggregates. In *ISLP*, pages 387–401, 1991.
[17] B. T. Loo, T. Condie, M. N. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative networking. *Commun. ACM*, 52(11):87–95, 2009.
[18] V. Markl. Breaking the chains: On declarative data analysis and data independence in the big data era. *PVLDB*, 7(13):1730–1733, 2014.
[19] M. Mazuran, E. Serra, and C. Zaniolo. Extending the power of datalog recursion. *VLDB J.*, 22(4):471–493, 2013.
[20] I. S. Mumick, H. Pirahesh, and R. Ramakrishnan. The magic of duplicates and aggregates. In *VLDB*, pages 264–277, 1990.
[21] C. H. Papadimitriou and M. Yannakakis. A note on succinct representations of graphs. *Information and Control*, 71(3):181–185, 1986.
[22] K. A. Ross and Y. Sagiv. Monotonic aggregation in deductive databases. *J. Comput. System Sci.*, 54(1):79–97, 1997.
[23] G. Sabidussi. The centrality index of a graph. *Psychometrika*, 31(4):581–603, 1966.
[24] J. S. Schlipf. The expressive powers of the logic programming semantics. *J. Comput. System Sci.*, 51(1):64–86, 1995.
[25] J. Seo, S. Guo, and M. S. Lam. SociaLite: An efficient graph query language based on datalog. *IEEE Trans. Knowl. Data Eng.*, 27(7):1824–1837, 2015.
[26] A. Shkapsky, M. Yang, M. Interlandi, H. Chiu, T. Condie, and C. Zaniolo. Big data analytics with datalog queries on Spark. In *SIGMOD*, pages 1135–1149, 2016.
[27] S. Sudarshan and R. Ramakrishnan. Aggregation and relevance in deductive databases. In *VLDB*, pages 501–511, 1991.
[28] A. Van Gelder. The well-founded semantics of aggregation. In *PODS*, pages 127–138, 1992.
[29] J. Wang, M. Balazinska, and D. Halperin. Asynchronous and fault-tolerant recursive datalog evaluation in shared-nothing engines. *PVLDB*, 8(12):1542–1553, 2015.
[30] M. Yang, A. Shkapsky, and C. Zaniolo. Scaling up the performance of more powerful datalog systems on multicore machines. *VLDB J.*, 26(2):229–248, 2017.
[31] C. Zaniolo, M. Yang, A. Das, A. Shkapsky, T. Condie, and M. Interlandi. Fixpoint semantics and optimization of recursive datalog programs with aggregates. *Th. Pract. Log. Program.*, 17(5-6):1048–1065, 2017.

Hardware-Conscious Stream Processing: A Survey

Shuhao Zhang[1], Feng Zhang[2], Yingjun Wu[3], Bingsheng He[1], Paul Johns[1]
[1]National University of Singapore, [2]Renmin University of China,
[3]Amazon Web Services

ABSTRACT

Data stream processing systems (DSPSs) enable users to express and run stream applications to continuously process data streams. To achieve real-time data analytics, recent researches keep focusing on optimizing the system latency and throughput. Witnessing the recent great achievements in the computer architecture community, researchers and practitioners have investigated the potential of adoption hardware-conscious stream processing by better utilizing modern hardware capacity in DSPSs. In this paper, we conduct a systematic survey of recent work in the field, particularly along with the following three directions: 1) computation optimization, 2) stream I/O optimization, and 3) query deployment. Finally, we advise on potential future research directions.

1 Introduction

A large volume of data is generated in real time or near real time and has grown explosively in the past few years. For example, IoT (Internet-of-Things) organizes billions of devices around the world that are connected to the Internet. IHS Markit forecasts [3] that 125 billion such devices will be in service by 2030, up from 27 billion in 2018. With the proliferation of such high-speed data sources, numerous data-intensive applications are deployed in real-world use cases exhibiting latency and throughput requirements, that can not be satisfied by traditional batch processing models. Despite the massive effort devoted to big data research, many challenges remain.

A data stream processing system (DSPS) is a software system which allows users to efficiently run stream applications that continuously analyze data in real time. For example, modern DSPSs [5, 6] can achieve very low processing latency in the order of milliseconds. Many research efforts are devoted to improving the performance of DSPSs from the research community [45, 23, 98, 92] and leading enterprises such as SAP [102], IBM [37], Google [9] and Microsoft [19]. Despite the success of the last several decades, more radical performance demand, complex analysis, as well as intensive state access in emerging stream applications [21, 91] pose great challenges to existing DSPSs. Meanwhile, significant achievements have been made in the computer architecture community, which has recently led to various investigations of the potential of *hardware-conscious DSPSs*, which aim to exploit the potential of accelerating stream processing on modern hardware [104, 98].

Fully utilizing hardware capacity is notoriously challenging. A large number of studies have been proposed in recent years [19, 66, 57, 58, 45, 98, 103, 104]. This paper hence aims at presenting a systematic review of prior efforts on hardware-conscious stream processing. Particularly, the survey is organized along with the following three directions: 1) computation optimization, 2) stream I/O optimization, and 3) query deployment. We aim to show what has been achieved and reveal what has been largely overlooked. We hope that this survey will shed light on the hardware-conscious design of future DSPSs.

2 Background

In this section, we introduce the common APIs and runtime architectures of modern DSPSs.

2.1 Common APIs

A DSPS needs to provide a set of APIs for users to express their stream applications. Most modern DSPSs such as Storm [6] and Flink [5] express a streaming application as a directed acyclic graph (DAG), where nodes in the graph represent operators, and edges represent the data dependency between operators. Figure 1 (a) illustrates the *word count* (WC) as an example application containing five operators. A detailed description of a few more stream applications can be found in [103].

Some earlier DSPSs (e.g., Storm [6]) require users to implement each operator manually. Recent efforts from Saber [45], Flink [5], Spark-Streaming [96], and Trident [55] aim to provide

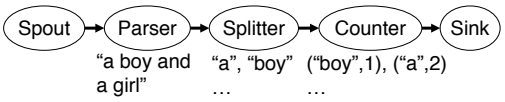

Figure 1: A stream processing example of word count.

declarative APIs (e.g., SQL) with rich built-in operations such as aggregation and join. Subsequently, many efforts have been devoted to improving the execution efficiency of the operations, especially by utilizing modern hardware (Section 4).

2.2 Common Runtime Architectures

Modern stream processing systems can be generally categorized based on their processing models including the Continuous Operator (CO) model and the Bulk Synchronous Parallel (BSP) model [87].

Continuous Operator Model: Under the CO model, the execution runtime treats each operator (a vertex of a DAG) as a single execution unit (e.g., a Java thread), and multiple operators communicate through message passing (an edge in a DAG). For scalability, each operator can be executed independently in multiple threads, where each thread handles a substream of input events with stream partitioning [43]. This execution model allows users to control the parallelism of each operator in a fine-grained manner [103]. This kind of design was adopted by many DSPSs such as Storm [6], Heron [49], Seep [17], and Flink [5] due to its advantage of low processing latency. Other recent hardware-conscious DSPSs adopt the CO model including Trill [19], BriskStream [104], and TerseCades [66].

Bulk-Synchronous Parallel Model: Under the BSP model, input stream is explicitly grouped into micro batches by a central coordinator and then distributed to multiple workers (e.g., a thread/machine). Subsequently, each data item in a micro batch is independently processed by going through the entire DAG (ideally by the same thread without any cross-operator communication). However, the DAG may contain synchronization barrier, where threads have to exchange their intermediate results (i.e., data shuffling). Taking WC as an example, the `Splitter` needs to ensure that the same word is always passed to the same thread of the `Counter`. Hence, a data shuffling operation is required before the `Counter`. As a result, such synchronization barriers break the DAG into multiple stages under the BSP model, and the communication between stages is managed by the central coordinator. This kind of design was adopted by Spark-streaming [96], Drizzle [87], and FlumeJava [18]. Other recent hardware-conscious DSPSs adopt the BSP model including Saber [45] and StreamBox [57].

Although there have been prior efforts to compare different models [82], it is still inconclusive that which model is more suitable for utilizing modern hardware – each model comes with its own advantages and disadvantages. For example, the BSP model naturally minimizes communication among operators inside the same DAG, but its single centralized scheduler has been identified with scalability limitation [87]. Moreover, its unavoidable data shuffling also brings significant communication overhead, as observed in recent research [104]. In contrast, CO model allows fine-grained optimization (i.e., each operator can be configured with different parallelisms and placements) but potentially incurs higher communication costs among operators. Moreover, the limitations of both models can potentially be addressed with more advanced techniques. For example, cross operator communication overhead (under both CO and BSP models) can be overcome by exploiting tuple batching [19, 103], high bandwidth memory [58, 70], data compression [66], InfiniBand [38] (Section 5), and architecture-aware query deployment [104, 98] (Section 6).

3 Survey Outline

The hardware architecture is evolving fast and provides a much higher processing capability than that traditional DSPSs were originally designed for. For example, recent *scale-up* servers can accommodate hundreds of CPU cores and terabytes of memory [4], providing abundant computing resources. Emerging network technologies such as Remote Direct Memory Access (RDMA) and 10Gb Ethernet significantly improve system ingress rate, making I/O no longer a bottleneck in many practical scenarios [57, 21]. However, prior studies [103, 98] have shown that existing data stream processing systems (DSPSs) severely underutilize hardware resources due to the unawareness of the underlying complex hardware architectures.

As summarized in Table 1, we are witnessing a revolution in the design of DSPSs that exploit emerging hardware capability, particularly along with the following three dimensions:

1) Computation Optimization: Contrary to conventional DBMSs, there are two key features in DSPSs that are fundamental to many stream applications and computationally expensive: *Windowing operation* [35] (e.g., windowing stream join) and *Out-of-order handling* [12]. The former

Table 1: Summary of the surveyed works

Research Dimensions	Key Concerns	Key Related Work
Computation Optimization	Synchronization overhead, work efficiency	CellJoin [23], FPGAJoin [47, 72], Handshake join [83, 73], PanJoin [65], HELLS-join [42, 41], Aggregation on GPU [45], Aggregation on FPGA [64, 24], Hammer Slide [84], StreamBox [57], Parallel Index Join [75]
Stream I/O Optimization	Time and space efficiency, data locality, and memory footprint	Batching [103], Stream with HBM [58, 70], TerseCades [66], Stream over InfiniBand [38], Stream on SSDs [51], and NVM-aware Storage [68]
Query Deployment	Operator interference, elastic scaling, and power constraint	Orchestrating [48, 22], StreamIt [16], SIMD [36], BitStream [8], Streams on Wires [60], HRC [88], RCM [89], CMGG [63], GStream [106], SABER [45], BriskStream [104]

deals with infinite stream, and the latter handles stream imperfection. The support for those expensive operations is becoming one of the major requirements for modern DSPSs and is treated as one of the key dimensions in differentiating modern DSPSs. Prior approaches use multicores [84, 57], heterogeneous architectures (e.g., GPUs and Cell processors) [23, 45], and Field Programmable Gate Arrays (FPGAs) [83, 73, 47, 72, 64, 24] for accelerating those operations.

2) Stream I/O Optimization: Cross-operator communication [103] is often a major source of overhead in stream processing. Recent work has revealed that the overhead due to cross-operator communication is significant, even without the TCP/IP network stack [103, 98]. Subsequently, research has been conducted on improving the efficiency of data grouping (i.e., output stream shuffling among operators) using High Bandwidth Memory (HBM) [58], compressing data in transmission with hardware accelerators and applying computation directly over compressed data [66], and leveraging InfiniBand for faster data flow [38]. Having said that, there are also cases where the application needs to temporarily store data for future usage [85] (i.e., state management [15]). Examples include stream processing with large window operation (i.e., workload footprint larger than memory capacity) and stateful stream processing with high availability (i.e., application states are kept persistently). To relieve the disk I/O overhead, recent work has investigated how to achieve more efficient state management, leveraging SSD [51] and non-volatile memory (NVM) [68].

3) Query Deployment: At an even higher point of view, researchers have studied launching a whole stream application (i.e., a query) into various hardware architectures. Similar to traditional database systems, the goal of query deployment in DSPS is to minimize operator interference/ cross-operator communication, balance hardware resource utilization, and so on. The major difference compared to traditional database systems lies in their different problem assumptions, and hence in their system architectures (e.g., infinite input stream [90], processing latency constraints [33], and unique cost function of streaming operators [102, 44]). To take advantage of modern hardware, prior works have exploited various hardware characteristics such as cache-conscious strategies [8], FPGA [60], and GPUs [88, 89, 63, 106]. Recent works have also looked into supporting hybrid architectures [45] and NUMA [104].

4 Computation Optimization

In this section, we review the literature on accelerating computationally expensive streaming operations using modern hardware.

4.1 Windowing Operation

In stream applications, the processing is mostly performed in the form of long-running queries known as continuous queries [11]. To handle potentially infinite data streams, continuous queries are typically limited to a window that limits the number of tuples to process at any point in time. The window can be defined based on the number of tuples (i.e., count based), function of time (i.e., time based) or sessions [86]. Window stream joins and window aggregation are two common expensive windowing operations in data stream processing.

4.1.1 Window Join

A common operation used in many stream analytical workloads is to *join* multiple data streams. Different from traditional join in relational databases [32], which processes a large batch of data at once, stream join has to produce results on the fly [39, 77, 31, 26]. By definition, the stream join operator performs over infinite streams. In practice, streams are cut into finite slices/windows [93]. In a two-way stream join, tuples from the left stream (`R`) are joined with tuples in the right stream (`S`) when the specified key attribute matches, and the timestamp of tuples from both streams falls within the same window.

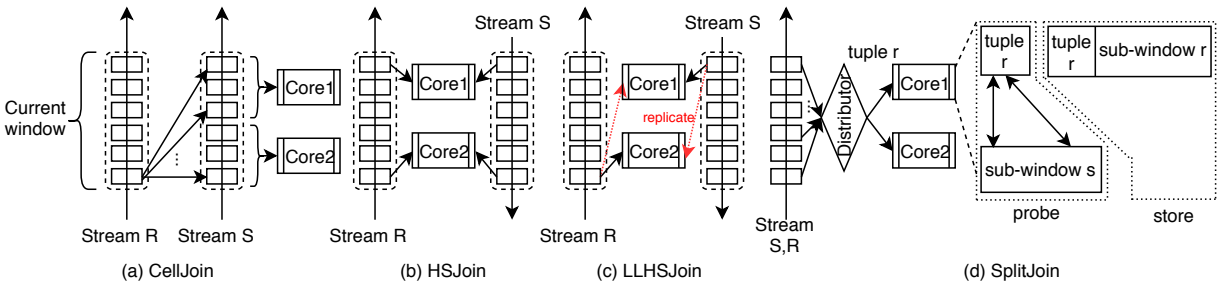

Figure 2: HW-conscious stream join algorithms (using two cores as an example).

Algorithm Overview. Kang et al. [39] described the first streaming join implementations. For each tuple r of stream R, 1) Scan the window associated with stream S and look for matching tuples; 2) Invalidate the old tuples in both windows; 3) Insert r into the window of R.

HW-Conscious Optimizations. The costly nature of stream join and the stringent response time requirements of stream applications have created significant interest in accelerating stream joining. Multicore processors that provide high processing capacity are ideal for executing costly windowed stream operators. However, fully exploiting the potential of a multicore processor is often challenging due to the complex processor microarchitecture, deep cache memory subsystem, and the unconventional programming model in general. Figure 2 illustrates the four representative studies on accelerating window stream joins described as follows.

CellJoin: An earlier work from Gedik et al. [23], called CellJoin, attempt to parallelize stream join on Cell processor, a heterogeneous multicore architecture. CellJoin generally follows Kang's [39] three-step algorithm. To utilize multicores, it re-partitions S, and each resulting partition is assigned to an individual core. In this way, the matching step can be performed in parallel on multiple cores. A similar idea has been adopted in the work by Karnagel et al. [42] to utilize the massively parallel computing power of GPU.

Handshake-Join (HSJoin): CellJoin essentially turns the join process into a scheduling and placement process. Subsequently, it is assumed that the window partition and fetch must be performed in global memory. The repartition and distribution mechanism essentially reveals that CellJoin generally follows the BSP model (see Section 2.2). This is later shown to be ineffective when the number of cores is large [83], and a new stream join technique called handshake join (i.e., HSJoin) was proposed. In contrast to CellJoin, HSJoin adopts the CO model. Specifically, both input streams notionally flow through the stream processing engine in opposite directions. As illustrated in Figure 2 (b), the two sliding windows are laid out side by side, and predicate evaluations are continuously performed along with the windows whenever two tuples encounter each other.

Low-Latency Handshake-Join (i.e., LLHSJoin): Despite its excellent scalability, the downside of HSJoin is that tuples may have to be queued for long periods of time before the match, resulting in high processing latency. In response, Roy et al. [73] propose a *low-latency* handshake-join (i.e., LLHSJoin) algorithm. The key idea is that, instead of sequentially forwarding each tuple through a pipeline of processing units, tuples are replicated and forwarded to all involved processing units (see the red dotted lines in Figure 2 (c)) before the join computation is carried out by one processing unit (called a home node).

SplitJoin: The state-of-the-art windowing join implementation called SplitJoin [62] parallelizes the join process via the CO model. Rather than forwarding tuples bidirectionally, as in HSJoin or LLHSJoin, SplitJoin broadcasts each newly arrived tuple t (from either S or R) to all processing units. In order to make sure that each tuple is processed only once, t is retained in exactly one processing unit chosen in a round-robin manner. Although SplitJoin [62] and HSJoin [83] can achieve the same concurrency theoretically without any central coordination, the former achieves a much lower latency due to the linear chaining delay of the HSJoin. While LLHSJoin [73] reduces the processing latency of HSJoin [83] by using a fast forwarding mechanism, it complicates the processing logic and reintroduces central coordination to the processing [62].

4.1.2 Window Aggregation

Another computationally heavy windowing operation is window aggregation, which summarizes the most recent information in a data stream. There are four workload characteristics [86] of stream

aggregation including 1) window type, which refers to the logic based on which system derives finite windows from a continuous stream, such as tumbling, sliding, and session; 2) windowing measures, which refers to ways to measure windows, such as time-based, count-based, and any other arbitrary advancing measures; 3) aggregate functions with different algebraic properties [81] such as *invertible*, *associative*, *commutative*, and *order-preserving*; and 4) stream (dis)order, which shall be discussed in Section 4.2.

Algorithm Overview. The trivial implementation is to perform the aggregation calculation from scratch for every arrived data. The complexity is hence $O(n)$, where n is the window size. Intuitively, efficiently leveraging previous calculation results for future calculation is the key to reducing computation complexity, which is often called incremental aggregation. However, the effectiveness of incremental aggregation depends heavily on the aforementioned workload characteristics such as the property of the aggregation function. For example, when the aggregation function is invertible (e.g., sum), we can simply update (i.e., increase) the aggregation results when a new tuple is inserted into the window and evict with the time complexity of $O(1)$. For faster answering median like function, which has to keep all the relevant inputs, instead of performing a sort on the window for each newly inserted tuple, one can maintain an *order statistics tree* as auxiliary data structure [34], which has $O(logn)$ worst-case complexity of its insertion, deletion, and rank function. Similarly, the reactive aggregator (RA) [80] with $O(logn)$ average complexity only works for aggregation function with the associative property. Those algorithms also differ from each other at their capability of handling different window types, windowing measures, and stream (dis)order [86]. Traub et al. [86] recently proposed a generalization of the stream slicing technique to handle different workload characteristics for window aggregation. It may be an interesting future work to study how the proposed technique can be applied to better utilize modern hardware architectures (e.g., GPUs).

HW-Conscious Optimizations. There are a number of works on accelerating windowing aggregation in a hardware-friendly manner. An early work by Mueller et al. [59] described implementation for a sliding windowed median operator on FPGAs. This is an operator commonly used to, for instance, eliminate noise in sensor readings and in data analysis tasks [71]. The algorithm skeleton adopted by the work is rather conventional: it first sorts elements within the sliding window and then computes the median. Compared to the $O(logn)$ complexity of using an order statistics tree as an auxiliary data structure [34], Mueller's method has a theoretically much higher complexity due to the sorting step ($O(nlogn)$). Nevertheless, their key contribution is on how the sorting and computing steps can be efficiently performed on FPGAs. Mueller's implementation [59] focuses on efficiently processing one sliding window without discussing how to handle subsequent sliding windows. Mueller et al. hence proposed conducting multiple computations for each sliding window by instantiating multiple aggregation modules concurrently [60].

Recomputing from scratch for each sliding window is costly, even if conducted in parallel [60]. Hence, a technique called pane [52] was proposed and later verified on FPGAs [64] to address this issue. The key idea is to divide overlapping windows into disjoint panes, compute sub-aggregates over each pane, and "roll up" the partial-aggregates to compute final results. Pane was later improved [46] and covers more cases (e.g., to support non-periodic windows [14]). However, the latest efforts are mostly theoretical, and little work has been done to validate the effectiveness of these techniques on modern hardware, e.g., FPGA and GPUs.

Saber [45] is a relational stream processing system targeting heterogeneous machines equipped with CPUs and GPUs. To achieve high throughput, Saber also adopts incremental aggregation computations utilizing the commutative and associative property of some aggregation functions such as count, sum, and average. Theodorakis et al. [84] recently studied the trade-off between workload complexity and CPU efficient streaming window aggregation. To this end, they proposed an implementation that is both workload- and CPU-efficient. Gong et al. [27] proposed an efficient and scalable accelerator based on FPGAs, called ShuntFlow, to support arbitrary window sizes for both reduce- and index-like sliding window aggregations. The key idea is to partition aggregation with extremely large window sizes into sub-aggregations with smaller window sizes that can enable more efficient use of FPGAs.

4.2 Out-of-Order Handling

In a real production environment, out-of-order[1] input data are not uncommon. A stream operator is considered to be *order-sensitive* if it requires input events to be processed in a certain predefined

[1]Other issues such as delay and missing can be seen as special cases of out-of-order.

order (e.g., chronological order). Handling out-of-order input data in an order-sensitive operator often turns out to be a performance bottleneck, as there is a fundamental conflict between data parallelism and order-sensitive processing – the former seeks to improve the throughput of an operator by letting more than one thread operate on different events concurrently, possibly out-of-order.

Algorithm Overview. Currently, there are three general techniques to be applied together with the order-sensitive operator to handle out-of-order data streams. The first utilizes a buffer-based data structure [12] that buffers incoming tuples for a while before processing. The key idea is to keep the data as long as possible (within the latency/buffer size constraint) to avoid out-of-order inputs. The second technique relies on punctuation [53], which is a special tuple in the event stream indicating the end of a substream. Punctuations guarantee that tuples are processed in monotonically increasing time sequence across punctuations, but not within the same punctuation. The third approach is to use speculative techniques [74]. The main idea is to process tuples without any delay, and recompute the results in the case of order violation. There are also techniques specifically designed for handling out-of-order in a certain type of operator such as window aggregation [86].

HW-Conscious Optimizations. Gulisano et al. [28] are among the first to handle out-of-order for high-performance stream join on multi-core CPUs. The proposed algorithm, called *scalejoin* is illustrated in Figure 3 (a). It first merges all incoming tuples into one stream (through a data structure called *scalegate*) and then distributes them to processing threads (PTs) to perform join. The output also needs to be merged and sorted before exiting the system. The use of the *scalegate* makes this work fall into the category of buffer-based approach and have inherent limitation of higher processing latency. Scalejoin has been implemented in FPGA [47] and further improved in another recent work [72]. They both found that the proposed system outperforms the corresponding fully optimized parallel software-based solution running on a high-end 48-core multiprocessor platform.

StreamBox [57] handles out-of-order event processing by the punctuation-based technique on multicore processors. Figure 3 (b) illustrates the basic idea of taking the stream join operator as an example. Relying on a novel data structure called *cascading container* to track dependencies between epochs (a group of tuples delineated by punctuation), StreamBox is able to maintain

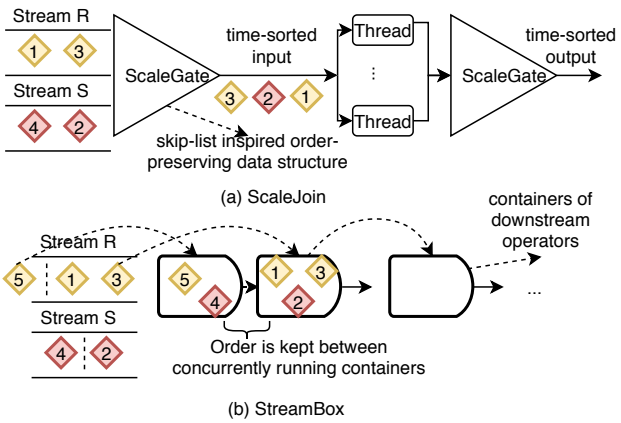

Figure 3: Multicore-friendly out-of-order handling.

the processing order among multiple concurrently executing containers that exploit the parallelism of modern multicore hardware.

Kuralenok et al. [50] attempt to balance the conflict between order-sensitive and multicore parallelism with an optimistic approach falling in the third approach. The basic idea is to conduct the joining process without any regulations, but apologize (i.e., sending amending signals) when the processing order is violated. They show that the performance of the proposed approach depends on how often reorderings are observed during runtime. In the case where the input order is naturally preserved, there is almost no overhead. However, it leads to extra network traffic and computations when reorderings are frequent. To apply such an approach to practical use cases, it is hence necessary to predict the probability of reordering, which could be an interesting future work.

4.3 Remarks

From the above discussion, it is clear that the key to accelerating windowing operators are mainly two folds. On the one hand, we should minimize the operation complexity. There are two common approaches: 1) incremental computation algorithms [45], which maximize reusing intermediate results, and 2) rely on efficient auxiliary data structures (e.g., indexing the contents of sliding window [95]) for reducing data (and/or instruction) accesses, especially cache misses. On the other hand, we should maximize the system concurrency [84]. This requires us to distribute workloads among multiple cores and minimize synchronization overhead among them [57]. Unfortunately, these optimization techniques are often at odds with each other. For example, incremental computation algorithm is complexity efficient but difficult to parallelize

due to inherent control dependencies in the CPU instruction [84]. Another example is that maintaining index structures for partial computing results may help to reduce data accesses, but it also brings maintenance overhead [54]. More investigation is required to better balance these conflicting aspects.

5 Stream I/O Optimization

In this section, we review the literature on improving the stream I/O efficiency using modern hardware.

5.1 Cross-operator Communication

Modern DSPSs [5, 6] are able to achieve very low processing latency in the order of milliseconds. However, excessive communication among operators [103] is still a key obstacle in further improving the performance of the DSPSs.

Kamburugamuve et al. [38] recently presented their findings on integrating Apache Heron [49] with InfiniBand and Intel OmniPath. The results show that both can be utilized to improve the performance of distributed streaming applications. Nevertheless, many optimization opportunities remain to be explored. For example, prior work [38] has evaluated Heron on InfiniBand with channel semantics but not remote direct memory access (RDMA) semantics [67]. The latter has shown to be very effective in other related works [76, 97].

Data compression is a widely used approach for reducing communication overhead. Pekhimenko et al. [66] recently examined the potential of using data compression in stream processing. Interestingly, they found that data compression does not necessarily lead to a performance gain. Instead, improvement can only be achieved through a combination of hardware accelerator (i.e., GPUs in their proposal) and new execution techniques (i.e., compute directly over compressed data).

As mentioned before, word count requires the same word to be transmitted to the same `Counter` operator (see Section 2.1). Subsequently, all DSPSs need to implement data grouping operations regardless of their processing model (i.e., the continuous operator model or bulk synchronous model). Data grouping involves excessive memory accesses that rely on hash-based data structures [103, 96]. Zeuch et al. [98] analyzed the design space of DSPSs optimized for modern multicore processors. In particular, they show that a queue-less execution engine based on query compilation, which replaces communication between operators with function calls, is highly suitable for modern hardware. Since data grouping can not be completely eliminated, they proposed a mechanism called "Upfront Partitioning with Late Merging", for efficient data grouping. Miao et al. [58] have exploited the possibility of accelerating data grouping using emerging 3D stacked memories such as high-bandwidth memory (HBM). By designing the system in a way that addresses the limited capacity of HBM and HBM's need for sequential-access and high parallelism, the resulting system can achieve several times of performance improvement over the baseline.

5.2 State Management

Emerging stream applications often require the underlying DSPS to maintain large application states so as to support complex real-time analytics [85, 15]. Representative example states required during stream processing include graph data structures [105] and transaction records [56].

The storage subsystem has undergone tremendous innovation in order to keep up with the ever-increasing performance demand. Wukong+S [105] is a recently proposed distributed streaming engine that provides real-time consistent query over streaming datasets. It is built based on Wukong [76], which leverages RDMA to optimize throughput and latency. Wukong+S also follows its pace to support stream processing while maintaining low latency and high throughput. Non-Volatile Memory (NVM) has emerged as a promising hardware and brings many new opportunities and challenges. Fernando et al. [68] has recently explored efficient approaches to support analytical workloads on NVM, where an NVM-aware storage layout for tables is presented based on a multidimensional clustering approach and a block-like structure to utilize the entire memory stack. As argued by the author, the storage structure designed on NVM may serve as the foundation for supporting features like transactional stream processing systems [29] in the future. Non-Volatile Memory Express (NVMe) -based solid-state devices (SSDs) are expected to deliver unprecedented performance in terms of latency and peak bandwidth. For example, the recently announced PCIe 4.0 based NVMe SSDs [1] are already capable of achieving a peak bandwidth of 4GB/s. Lee et al. [51] have recently investigated the performance limitations of current DSPSs on managing application states on SSDs and have shown that query-aware optimization can significantly improve the performance of stateful stream processing on SSDs.

5.3 Remarks

Hardware-conscious stream I/O optimization is still in its early days. Most prior work attempts at

mitigating the problem through a purely software approach, such as I/O-aware query deployment [94]. The emerging hardware such as Non-Volatile Memory (NVM) and InfiniBand with RDMA open up new opportunities for further improving stream I/O performance [105]. Meanwhile, the usage of emerging hardware accelerators such as GPUs further brings new opportunities to trade-off computation and communication overhead [66]. However, a model-guided approach to balance the trade-off is still generally missing in existing work. We hence expect more work to be done in this direction in the near future.

6 Query Deployment

We now review prior works from a higher level of abstraction, the query/application dimension. We summarize them based on their deployment targets: multicore CPUs, GPUs, and FPGAs.

6.1 Multicore Stream Processing

Language and Compiler. Multicore architectures have become ubiquitous. However, programming models and compiler techniques for employing multicore features are still lagging behind hardware improvements. Kudlur et al. [48] were among the first to develop a compiler technique to map stream application to a multicore processor. By taking the Cell processor as an example, they study how to compile and run a stream application expressed in their proposed language, called `StreamIt`. The compiler works in two steps: 1) operator fission optimization (i.e., split one operator into multiple ones) and 2) assignment optimization (i.e., assign each operator to a core). The two-step mapping is formulated as an integer linear programming (ILP) problem and requires a commercial ILP solver. Noting its NP-Hardness, Farhad et al. [22] later presented an approximation algorithm to solve the mapping problem. Note that the mapping problem from Kudlur et al. [48] considers only CPU loads and ignores communications bandwidth. In response, Carpenter et al. [16] developed an algorithm that maps a streaming program onto a heterogeneous target, further taking communication into consideration. To utilize a SIMD-enabled multicore system, Hormati et al. [36] proposed vectorizing stream applications. Relying on high-level information, such as the relationship between operators, they were able to achieve better performance than general vectorization techniques. Agrawal et al. [8] proposed a cache conscious scheduling algorithm for mapping stream application on multicore processors. In particular, they developed the theoretical lower bounds on cache misses when scheduling a streaming pipeline on multiple processors, and the upper bound of the proposed cache-based partitioning algorithm called *seg_cache*. They also experimentally found that scheduling solely based on the cache effects can often be more effective than the conventional load-balancing (based on computation cost) approaches.

Multicore-aware DSPSs. Recently, there has been a fast growing amount of interest in building multicore-friendly DSPSs. Instead of statically compiling a program as done in `StreamIt` [48, 22, 16], these DSPSs provide better elasticity for application execution. They also allow the usage of general-purpose programming languages (e.g., Java, Scala) to express stream applications. Tang et al. [79] studied the data flow graph to explore the potential parallelism in a DSPS and proposed an *auto-pipelining* solution that can utilize multicore processors to improve the throughput of stream processing applications. For economic reasons, power efficiency has become more and more important in recent years, especially in the HPC domains. Kanoun et al. [40] proposed a multicore scheme for stream processing that takes power constraints into consideration. Trill [19] is a single-node query processor for temporal or streaming data. Contrary to most distributed DSPSs (e.g., Storm, Flink) adopting the continuous operator model, Trill runs the whole query only on the thread that feeds data to it. Such an approach has shown to be especially effective [98] when applications contain no synchronization barriers.

6.2 GPU-Enabled Stream Processing

GPUs are the most popular heterogeneous processors due to their high computing capacity. However, due to their unconventional execution model, special designs are required to efficiently adapt stream processing to GPUs.

Single-GPU. Verner et al. [88] presented a general algorithm for processing data streams with real-time stream scheduling constraints on GPUs. This algorithm assigns data streams to CPUs and GPUs based on their incoming rates. It tries to provide an assignment that can satisfy different requirements from various data streams. Zhang et al. [100] developed a holistic approach to building DSPSs using GPUs. They design a latency-driven GPU-based framework, which mainly focuses on real-time stream processing. Due to the limited memory capacity of GPUs, the window size of the stream operator plays an important role in system performance. Pinnecke et al. [69] studied the influence of window size and proposed a partitioning

method for splitting large windows into different batches, considering both time and space efficiency. SABER [45] is a window-based hybrid stream processing framework aiming to utilize CPUs and GPUs concurrently.

Multi-GPU. Multi-GPU systems provide tremendous computation capacity, but also pose challenges like how to partition or schedule workloads among GPUs. Verner et al. [89] extend their method [88] to a single node with multiple GPUs. A scheduler controls stream placement and guarantees that the requirements among different streams can be met. GStream [106] is the first data streaming framework for GPU clusters. GStream supports stream processing applications in the form of a C++ library; it uses MPI to implement the data communication between different nodes and uses CUDA to conduct stream operations on GPUs. Alghabi et al. [10] first introduced the concept of stateful stream data processing on a node with multiple GPUs. Nguyen et al. [63] considered the scalability with the number of GPUs on a single node, and developed a GPU performance model for stream workload partitioning in multi-GPU platforms with high scalability. Chen et al. [20] proposed G-Storm, which enables Storm [6] to utilize GPUs and can be applied to various applications that Storm has already supported.

6.3 FPGA-Enabled Stream Processing

FPGAs are programmable integrated circuits whose hardware interconnections can be configured by users. Due to their low latency, high energy efficiency, and low hardware engineering cost, FPGAs have been explored in various application scenarios, including stream processing.

Hagiescu et al. [30] first elaborated challenges to implementing stream processing on FPGAs and proposed algorithms that optimize processing throughput and latency for FPGAs. Mueller et al. [60] provided *Glacier*, which is an FPGA-based query engine that can process queries on streaming data from networks. The operations in *Glacier* include selection, aggregation, grouping, and windows. Experiments show that using FPGAs helps achieve much better performance than using conventional CPUs. A common limitation of an FPGA-based system is its expensive synthesis process, which takes a significant time to compile the application into hardware designs for FPGAs. This makes FPGA-based systems inflexible in adapting to query changes. In response, Najafi et al. [61] demonstrated Flexible Query Processor (FQP), an online reconfigurable event stream query processor that can accept new queries without disrupting other queries in execution.

6.4 Remarks

Existing systems usually involve heterogeneous processors along with CPUs. Such heterogeneity opens up both new opportunities and poses challenges for scaling stream processing. From the above discussion, it is clear that both GPUs and FPGAs have been successfully applied for scaling up stream processing. FPGAs have low latency and are hardware configurable. Hence, they are suitable for special application scenarios, such as a streaming network.

7 System Design Requirements

In 2005, Stonebraker et al. [78] outlines eight requirements of real-time data stream processing. Since then, tremendous improvements have been made thanks to the great efforts from both industry and the research community. We now summarize how hardware-conscious optimization techniques mitigate the gap between DSPSs and requirements while highlighting the insufficiency.

Most DSPSs are designed with the principle of "*Keep the Data Moving*" [78], and hence aim to process input data "on-the-fly" without storing them. As a result, message passing is often a key component in the current DSPSs. To mitigate the overhead, researchers have recently attempted to improve the cross-operator communication efficiency by taking advantage of the latest advancement in network infrastructure [38], compression using hardware accelerator [66], and efficient algorithms by exploiting new hardware characteristics [58]. Going forward, we expect more work to be done for hardware-conscious stream I/O optimization.

Handling out-of-order input streams is relevant to both the *Handle Stream Imperfections* and *Generate Predictable Outcomes* [78] requirements. In real-time stream systems where the input data are not stored, the infrastructure must make provision for handling data that arrive late or are delayed, missing or out-of-sequence. Correctness can be guaranteed in some applications only if time-ordered and deterministic processing is maintained throughout the entire processing pipeline. Despite the significant efforts, existing DSPSs are still *far from ideal* for exploiting the potential of modern hardware. For example, as observed in a recent work [104], the same DSPS (i.e., StreamBox) delivers much lower throughput on modern multicore processors as a result of enabling ordering guarantees.

The state management in DSPSs is more

related to the *Integrate Stored and Streaming Data* [78] requirement. For many stream processing applications, comparing the "present" with the "past" is a common task. Thus, the system must provide careful management of the stored states. However, we observe that only a few related studies attempt to improve state management efficiency levering modern hardware [51]. There are still many open questions to be resolved, such as new storage formats, indexing techniques for emerging hardware architectures and applications [29, 101]. New media applications such as live audio streaming services [91] also challenge existing systems in terms of new processing paradigms.

The *Partition and Scale Applications Automatically* [78] requires a DSPS to be able to elastically scale up and down in order to process input streams with varying characteristics. However, based on our analysis, little work has considered scaling *down* the processing efficiently (and easily scaling up later) in a hardware-aware manner. A potential direction is adopting a server-less computing paradigm [13] with the help of novel memory techniques such as Non-Volatile Memory (NVM) into DSPSs. However, questions such as how to efficiently manage the partial computing state in GPUs or FPGAs still remain unclear.

The proliferation of high-rate data sources has accelerated the development of next-generation performance-critical DSPSs. For example, the new 5G network promises blazing speeds, massive throughput capability, and ultra-low latencies [2], thus bringing the higher potential for performance critical stream applications. In response, high-throughput stream processing is essential to keeping up with data streams in order to satisfy the *Process and Respond Instantaneously* [78] requirement. However, achieving high-throughput stream processing is challenging, especially when expensive windowing operations are deployed. By better utilizing modern hardware, researchers and practitioners have achieved promising results. For example, SABER processes 79 million tuples per second with eight CPU cores for Yahoo Streaming Benchmark, outperforming other DSPSs several times [7]. Nevertheless, current results also show that there is still room for improvement on a single node, and this constitutes an opportunity for designing the next-generation DSPSs [99].

Two requirements including *Query using SQL on Streams* and *Guarantee Data Safety and Availability* are overlooked by most existing HW-conscious optimization techniques in DSPSs. In particular, how to design HW-aware SQL statements for DSPSs, and how best to guarantee data safety and system availability when adopting modern hardware, such as NVM for efficient local backup and high-speed network for remote backup, remain an open question.

8 Conclusion

In this paper, we have discussed relevant literature from the field of hardware-conscious DSPSs, which aim to utilize modern hardware capabilities for accelerating stream processing. Those works have significantly improved DSPSs to better satisfy the design requirements raised by Stonebraker et al. [78]. In the following, we list some additional advice on future research directions.

Scale-up and -out Stream Processing. As emphasized by Gibbons [25], scaling both out and up is crucial to effectively improving the system performance. In situ analytics enable data processing at the point of data origin, thus reducing the data movements across networks; Powerful hardware infrastructure provides an opportunity to improve processing performance within a single node. To this end, many recent works have exploited the potential of high-performance stream processing on a single node [45, 57, 98]. However, the important question of how best to use powerful local nodes in the context of large distributed computation setting still remains unclear.

Stream Processing Processor. With the wide adoption of stream processing today, it may be a good time to revisit the design of a specific processor for DSPSs. GPUs [45] provide much higher bandwidth than CPUs, but it comes with larger latency as tuples must be first accumulated in order to fully utilize thousands of cores on GPU; FPGA [47] has its advantage in providing low latency, low power consumption computation but its throughput is still much lower compared to GPUs. The requirement for an ideal processor for stream processing includes *low latency*, *low power consumption*, and *high bandwidth*. On the other hand, components like complex control logic may be sacrificed as stream processing logic is usually predefined and fixed. Further, due to the nature of continuous query processing, it is ideal to keep the entire instruction set close to processor [103].

Acknowledgments. The authors would like to thank the anonymous reviewer and the associate editor, Pınar Tözün, for their insightful comments on improving this manuscript. This work is supported by a MoE Tier 1 grant (T1 251RES1824) and a MoE Tier 2 grant (MOE2017-T2-1-122) in Singapore. Feng Zhang's work was partially supported by the National Natural Science Foundation of China (Grant No. 61802412, 61732014).

9 References

[1] Corsair force series nvme ssd. https://hothardware.com/news/corsair-mp600.

[2] Ericsson mobility report november 2018, https://www.ericsson.com/assets/local/mobility-report/documents/2018/ericsson-mobility-report-november-2018.pdf.

[3] Number of connected iot devices will surge to 125 billion by 2030, ihs markit says. https://en.ctimes.com.tw/DispNews.asp?O=HK1APAPZ546SAA00N9.

[4] Sgi uv 300, https://www.earlham.ac.uk/sgi-uv300, 2015.

[5] Apache flink, https://flink.apache.org/, 2018.

[6] Apache storm, http://storm.apache.org/, 2018.

[7] Do we need distributed stream processing? https://lsds.doc.ic.ac.uk/blog/do-we-need-distributed-stream-processing, 2019.

[8] K. Agrawal and et al. Cache-conscious scheduling of streaming applications. In *SPAA*, 2012.

[9] T. Akidau and et al. Millwheel: Fault-tolerant stream processing at internet scale. *Proc. VLDB Endow.*, 2013.

[10] F. Alghabi and et al. A scalable software framework for stateful stream data processing on multiple gpus and applications. In *GPU Computing and Applications*. 2015.

[11] A. Arasu and et al. The cql continuous query language: Semantic foundations and query execution. *The VLDB Journal*, 2006.

[12] S. Babu and et al. Exploiting k-constraints to reduce memory overhead in continuous queries over data streams. *ACM Trans. Database Syst.*, 2004.

[13] I. Baldini and et al. Serverless computing: Current trends and open problems. *Research Advances in Cloud Computing*, 2017.

[14] P. Carbone and et al. Cutty: Aggregate sharing for user-defined windows. In *CIKM*, 2016.

[15] P. Carbone and et al. State management in apache flink: Consistent stateful distributed stream processing. *Proc. VLDB Endow.*, 2017.

[16] P. M. Carpenter, A. Ramirez, and E. Ayguade. Mapping stream programs onto heterogeneous multiprocessor systems. In *CASES*, 2009.

[17] R. Castro Fernandez and et al. Integrating Scale out and Fault Tolerance in Stream Processing Using Operator State Management. In *SIGMOD*, 2013.

[18] C. Chambers and et al. Flumejava: Easy, efficient data-parallel pipelines. In *PLDI*, 2010.

[19] B. Chandramouli and et al. Trill: A high-performance incremental query processor for diverse analytics. Proc. VLDB Endow., Aug. 2015.

[20] Z. Chen and et al. G-Storm: GPU-enabled high-throughput online data processing in Storm. In *Big Data*, 2015.

[21] J. A. Colmenares and et al. Ingestion, indexing and retrieval of high-velocity multidimensional sensor data on a single node. volume abs/1707.00825, 2017.

[22] S. M. Farhad and et al. Orchestration by approximation: Mapping stream programs onto multicore architectures. In *ASPLOS*, 2011.

[23] B. Gedik and et al. Celljoin: A parallel stream join operator for the cell processor. *The VLDB Journal*, 2009.

[24] P. R. Geethakumari and et al. Single window stream aggregation using reconfigurable hardware. In *ICFPT*, 2017.

[25] P. B. Gibbons. Big data: Scale down, scale up, scale out. In *IPDPS*, 2015.

[26] L. Golab and M. T. Özsu. Processing sliding window multi-joins in continuous queries over data streams. In *Proc. VLDB Endow.*, 2003.

[27] S. Gong and et al. Shuntflow: An efficient and scalable dataflow accelerator architecture for streaming applications. In *ADAC*, 2019.

[28] V. Gulisano and et al. Scalejoin: A deterministic, disjoint-parallel and skew-resilient stream join. In *Big Data*, 2015.

[29] P. Götze and et al. Query planning for transactional stream processing on heterogeneous hardware: Opportunities and limitations. In *BTW 2019*, 2019.

[30] A. Hagiescu and et al. A computing origami: folding streams in FPGAs. In *DAC*, 2009.

[31] M. A. Hammad and et al. Stream window join: tracking moving objects in sensor-network databases. In *SSDM*, 2003.

[32] J. He and et al. Revisiting co-processing for hash joins on the coupled cpu-gpu architecture. *Proc. VLDB Endow.*, 2013.

[33] T. Heinze and et al. Latency-aware Elastic Scaling for Distributed Data Stream Processing Systems. *TPDS*, 2014.

[34] M. Hirzel and et al. Spreadsheets for stream processing with unbounded windows and partitions. In *DEBS*, 2016.

[35] M. Hirzel and et al. Sliding-window aggregation algorithms: Tutorial. In *DEBS*, 2017.

[36] A. H. Hormati and et al. Macross: Macro-simdization of streaming applications. In *ASPLOS*, 2010.

[37] N. Jain and et al. Design, implementation, and evaluation of the linear road benchmark on the stream processing core. In *SIGMOD*, 2006.

[38] S. Kamburugamuve and et al. Low latency stream processing: Apache heron with infiniband & intel omni-path. In *UCC*, 2017.

[39] J. Kang and et al. Evaluating window joins over unbounded streams. In *ICDE*, 2003.

[40] K. Kanoun and et al. Low power and scalable many-core architecture for big-data stream computing. In *VLSI*, 2014.

[41] T. Karnagel and et al. The hells-join: A heterogeneous stream join for extremely large windows. In *DaMoN*, 2013.

[42] T. Karnagel and et al. Stream join processing on heterogeneous processors. In *BTW Workshops*, 2013.

[43] N. R. Katsipoulakis, A. Labrinidis, and P. K. Chrysanthis. A holistic view of stream partitioning costs. *Proc. VLDB Endow.*, 2017.

[44] I. Kolchinsky and A. Schuster. Join query optimization techniques for complex event processing applications. *Proc. VLDB Endow.*, 2018.

[45] A. Koliousis and et al. Saber: Window-based hybrid stream processing for heterogeneous architectures. In *SIGMOD*, 2016.

[46] S. Krishnamurthy and et al. On-the-fly sharing for streamed aggregation. In *SIGMOD*, 2006.

[47] C. Kritikakis and et al. An fpga-based high-throughput stream join architecture. In *FPL*, Aug. 2016.

[48] M. Kudlur and S. Mahlke. Orchestrating the execution of stream programs on multicore platforms. *SIGPLAN Not.*, 2008.

[49] S. Kulkarni and et al. Twitter heron: Stream processing at scale. In *SIGMOD*, 2015.

[50] I. E. Kuralenok and et al. An optimistic approach to handle out-of-order events within analytical stream processing. In *CEUR Workshop*, 2018.

[51] G. Lee and et al. High-performance stateful stream processing on solid-state drives. In *APSys*, 2018.

[52] J. Li and et a. No pane, no gain: Efficient evaluation of sliding-window aggregates over data streams. *SIGMOD Rec.*, 2005.

[53] J. Li and et al. Out-of-order processing: A new architecture for high-performance stream systems. *Proc. VLDB Endow.*, 2008.
[54] Q. Lin and et al. Scalable distributed stream join processing. In *SIGMOD*, 2015.
[55] N. Marz. Trident API Overview. `github.com/nathanmarz/storm/wiki/Trident-APIOverview`.
[56] J. Meehan and et al. S-store: streaming meets transaction processing. *Proc. VLDB Endow.*, 2015.
[57] H. Miao and et al. Streambox: Modern stream processing on a multicore machine. In *USENIX ATC*, 2017.
[58] H. Miao and et al. Streambox-hbm: Stream analytics on high bandwidth hybrid memory. *arXiv preprint arXiv:1901.01328*, 2019.
[59] R. Mueller and et al. Data processing on fpgas. *Proc. VLDB Endow.*, 2009.
[60] R. Mueller and et al. Streams on wires: a query compiler for FPGAs. *Proc. VLDB Endow.*, 2009.
[61] M. Najafi and et al. Flexible query processor on FPGAs. *Proc. VLDB Endow.*, 2013.
[62] M. Najafi and et al. Splitjoin: A scalable, low-latency stream join architecture with adjustable ordering precision. In *USENIX ATC*, 2016.
[63] D. Nguyen and J. Lee. Communication-aware mapping of stream graphs for multi-gpu platforms. In *CGO*, 2016.
[64] Y. Oge, M. Yoshimi, and et al. An efficient and scalable implementation of sliding-window aggregate operator on fpga. In *CANDAR*, 2013.
[65] F. Pan and H. Jacobsen. Panjoin: A partition-based adaptive stream join. *CoRR*, abs/1811.05065, 2018.
[66] G. Pekhimenko and et al. Tersecades: Efficient data compression in stream processing. In *USENIX ATC*, 2018.
[67] G. F. Pfister. An introduction to the infiniband architecture. *High Performance Mass Storage and Parallel I/O*, 42:617–632, 2001.
[68] G. Philipp and et al. An nvm-aware storage layout for analytical workloads. In *ICDEW*, 2018.
[69] M. Pinnecke and et al. Toward GPU Accelerated Data Stream Processing. In *GvD*, 2015.
[70] C. Pohl. Stream processing on high-bandwidth memory. In *Grundlagen von Datenbanken*, 2018.
[71] L. Rabiner and et al. Applications of a nonlinear smoothing algorithm to speech processing. *TASSP*, 1975.
[72] C. Rousopoulos and et al. A generic high throughput architecture for stream processing. In *FPL*, 2017.
[73] P. Roy and et al. Low-latency handshake join. *Proc. VLDB Endow.*, May 2014.
[74] E. Ryvkina and et al. Revision processing in a stream processing engine: A high-level design. In *ICDE*, 2006.
[75] A. Shahvarani and H.-A. Jacobsen. Parallel index-based stream join on a multicore cpu. `https://arxiv.org/pdf/1903.00452.pdf`, 2019.
[76] J. Shi and et al. Fast and concurrent rdf queries with rdma-based distributed graph exploration. In *OSDI*, 2016.
[77] U. Srivastava and J. Widom. Memory-limited execution of windowed stream joins. In *Proc. VLDB Endow.*, 2004.
[78] M. Stonebraker and et al. The 8 requirements of real-time stream processing. *SIGMOD Rec.*, 2005.
[79] Y. Tang and B. Gedik. Autopipelining for data stream processing. *TPDS*, 2013.
[80] K. Tangwongsan and et al. General incremental sliding-window aggregation. *Proc. VLDB Endow.*, 2015.
[81] K. Tangwongsan and et al. *Sliding-Window Aggregation Algorithms*. Springer International Publishing, 2018.
[82] W. B. Teeuw and H. M. Blanken. Control versus data flow in parallel database machines. *TPDS*, 1993.
[83] J. Teubner and R. Mueller. How soccer players would do stream joins. In *SIGMOD*, 2011.
[84] G. Theodorakis and et al. Hammer slide: work-and cpu-efficient streaming window aggregation. In *ADMS*, 2018.
[85] Q.-C. To and et al. A survey of state management in big data processing systems. *The VLDB Journal*, 2018.
[86] J. Traub and et al. Efficient window aggregation with general stream slicing. In *EDBT*, pages 97–108, 2019.
[87] S. Venkataraman and et al. Drizzle: Fast and adaptable stream processing at scale. In *SOSP*, 2017.
[88] U. Verner and et al. Processing data streams with hard real-time constraints on heterogeneous systems. In *ICS*, 2011.
[89] U. Verner and et al. Scheduling processing of real-time data streams on heterogeneous multi-gpu systems. In *SYSTOR*, 2012.
[90] S. D. Viglas and J. F. Naughton. Rate-based query optimization for streaming information sources. In *SIGMOD*, 2002.
[91] Z. Wen and et al. Rtsi: An index structure for multi-modal real-time search on live audio streaming services. In *ICDE*, 2018.
[92] Y. Wu and K. Tan. Chronostream: Elastic stateful stream computation in the cloud. In *ICDE*, Apr. 2015.
[93] J. Xie and J. Yang. A survey of join processing in data streams. *Data Streams: Models and Algorithms*, 2007.
[94] J. Xu and et al. T-storm: Traffic-aware online scheduling in storm. In *ICDCS*, 2014.
[95] Y. Ya-xin and et al. An indexed non-equijoin algorithm based on sliding windows over data streams. *Wuhan University Journal of Natural Sciences*, 2006.
[96] M. Zaharia and et al. Discretized streams: Fault-tolerant streaming computation at scale. In *SOSP*, 2013.
[97] E. Zamanian and et al. The end of a myth: Distributed transactions can scale. *Proc. VLDB Endow.*, 2017.
[98] S. Zeuch and et al. Analyzing efficient stream processing on modern hardware. *Proc. VLDB Endow.*, 2019.
[99] S. Zeuch and et al.l. The nebulastream platform: Data and application management for the internet of things. In *CIDR*, 2020.
[100] K. Zhang and et al. A holistic approach to build real-time stream processing system with gpu. *JPDC*, 2015.
[101] S. Zhang and et al. Towards concurrent stateful stream processing on multicore processors, `https://arxiv.org/abs/1904.03800`.
[102] S. Zhang and et al. Multi-query optimization for complex event processing in sap esp. In *ICDE*, 2017.
[103] S. Zhang and et al. Revisiting the design of data stream processing systems on multi-core processors. In *ICDE*, 2017.
[104] S. Zhang and et al. Briskstream: Scaling Data Stream Processing on Multicore Architectures. In *SIGMOD*, 2019.
[105] Y. Zhang and et al. Sub-millisecond stateful stream querying over fast-evolving linked data. In *SOSP*, 2017.
[106] Y. Zhang and F. Mueller. Gstream: A general-purpose data streaming framework on gpu clusters. In *ICPP*, 2011.

Domain- and Structure-Agnostic End-to-End Entity Resolution with JedAI

George Papadakis[1], Leonidas Tsekouras[2], Emmanouil Thanos[3], George Giannakopoulos[2], Themis Palpanas[4], Manolis Koubarakis[1]
[1]National and Kapodistrian University of Athens, Greece {gpapadis,koubarak}@di.uoa.gr
[2]NCSR "Demokritos", Greece {ltsekouras, ggianna}@iit.demokritos.gr
[3]KU Leuven, Belgium emmanouil.thanos@kuleuven.be
[4]Paris Descartes University, France themis@mi.parisdescartes.fr

ABSTRACT

We present JedAI, a new open-source toolkit for end-to-end Entity Resolution. JedAI is *domain-agnostic* in the sense that it does not depend on background expert knowledge, applying seamlessly to data of any domain with minimal human intervention. JedAI is also *structure-agnostic*, as it can process any type of data, ranging from structured (relational) to semi-structured (RDF) and un-structured (free-text) entity descriptions. JedAI consists of two parts: *(i)* JedAI-core is a library of numerous state-of-the-art methods that can be mixed and matched to form (thousands of) end-to-end workflows, allowing for easily benchmarking their relative performance. *(ii)* JedAI-gui is a user-friendly desktop application that facilitates the composition of complex workflows via a wizard-like interface. It is suitable for both lay and power users, offering concrete guidelines and automatic configuration, as well as manual configuration options, visual exploration, and detailed statistics for each method's performance. In this paper, we also delve into the new features of JedAI's latest version (2.1), and demonstrate its performance experimentally.

1. INTRODUCTION

Entity Resolution (ER) aims to detect different entity profiles that describe the same real-world objects [4]. It is a core task for data integration, with many applications that range from knowledge bases to question answering [6]. Yet, the functionality of the available ER systems is significantly restricted by the format of the various data collections. We can actually distinguish the existing systems into those crafted for *structured* (relational) data that is described by a well-defined schema, and those applying exclusively to *semi-structured data* that is associated with loose, diverse schemata and resides in XML/RDF repositories or SPARQL endpoints.

The latter category encompasses Link Discovery frameworks, which are surveyed in [20]. LIMES[1] and Silk[2] are the most prominent representatives. However, most of these tools implement only the method(s) introduced by their creators, and/or are suitable for power users, requiring the manual configuration of matching rules, or a labeled dataset for learning such rules in a supervised way [14]. Another drawback is that none of them is applicable to structured data, while half of them lack a GUI [20].

A larger variety of tools is available for structured data. A thorough list of 15 commercial and 18 non-commercial systems (such as Febrl[3] and Dedoop[4]) is analyzed in [15]. Most of them, though, suffer from one or more of the following problems: they cover the ER pipeline partially, they constitute stand-alone systems with a limited variety of methods, or they are exclusively meant for power users, providing insufficient guidelines on how to perform ER efficiently and effectively [15]. Magellan [16] resolves these issues, offering various blocking and matching methods. However, it is restricted to Record Linkage over relational data, lacks a GUI (it merely offers a command-line interface) and requires heavy user involvement; its goal is actually to facilitate the development of tailor-made methods for the data at hand. Similarly, heavy user involvement is required by the crowd-sourcing systems Corleone [11] and Falcon [5], which also address ER in an end-to-end manner through various efficiency techniques.

To overcome these drawbacks, we developed the **Java gEneric DAta Integration**[5] toolkit (JedAI for short), aiming to facilitate researchers, practitioners and lay users in applying ER solutions to any type of data. At its core lies the *end-to-end ER workflow* of Figure 1, which covers both Dirty ER (Deduplication) and Clean-Clean ER (Record Linkage). JedAI conveys one of the largest libraries

[1]http://aksw.org/Projects/LIMES.html

[2]http://silkframework.org
[3]https://sourceforge.net/projects/febrl
[4]https://dbs.uni-leipzig.de/dedoop
[5]http://jedai.scify.org

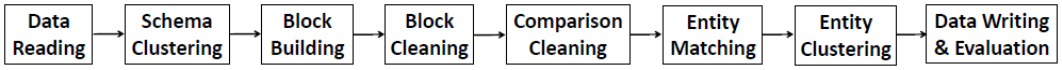

Figure 1: The end-to-end workflow for Entity Resolution implemented by JedAI.

with state-of-the-art ER methods, while being one of the few ER systems that is suitable even for lay users, providing an intuitive GUI. In more detail, JedAI has the following advantages:

1) *Structure-agnostic functionality.* JedAI applies uniformly to structured, semi-structured and unstructured (free-text) data.

2) *Domain-agnostic functionality.* All methods implemented by JedAI apply to data from any domain, ranging from homogeneous census, customer, product and bibliographic data to heterogeneous Knowledge Bases. The only requirement is that their entities contain string-dominated values.

3) *High time efficiency.* JedAI offers the largest variety of blocking and block processing methods. No other toolkit exploits the benefits of schema-agnostic blocking, which minimizes user involvement, while maximizing recall [22]. Also, no other toolkit includes the Block and Comparison Cleaning steps, which are indispensable for enhancing the time efficiency of ER by orders of magnitude [26]. JedAI also uses GNU Trove[6] for minimizing its memory footprint. This is done by operating on primitive data types instead of objects. E.g., collections of integer values are handled through the 4-byte `int` type instead of the 16-byte `Integer` objects, thus occupying up to 75% less memory. This also reduces the running time by more than 50% when compared to native Java [28].

4) *Hands-off functionality.* JedAI couples every implemented method with a default configuration of its internal parameters. Thus, neither manual parameter fine-tuning nor expert knowledge are required for building an end-to-end ER workflow; users simply select one or more methods per step.

5) *Learning-free functionality.* None of the implemented methods requires a labelled dataset for its training. Their default configuration renders them directly applicable to any data. Labelled data in the form of all true matches in a dataset (i.e., positive instances) are only required for evaluating the performance of a method or workflow and for fine-tuning its configuration parameters. In contrast, the learning-based methods require a labelled dataset for their operation, i.e., in order to learn their blocking or matching model. This labelled dataset includes not only the positive instances considered by JedAI, but also a carefully selected sample of non-matches (i.e., negative instances). The relative number of positive and negative instances as well as their representativity affects significantly the performance of the learned model. No such restrictions apply to JedAI's learning-free methods.

[6] `https://bitbucket.org/trove4j/trove`

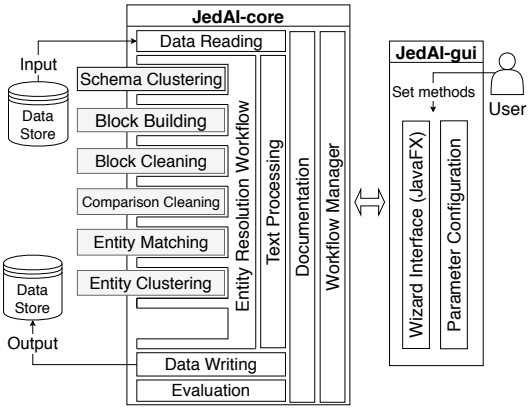

Figure 2: JedAI's architecture.

Most importantly, JedAI's learning-free methods optimize their performance by fine-tuning their internal parameters, which are generic in the sense that they are independent of the data at hand. In contrast, the features of learning-based methods are domain or dataset-specific, typically requiring heavy human intervention for their definition. Note also that JedAI's learning-free methods are inherently crafted for highly noisy and heterogeneous data [24]: to address possible errors in attribute values, their domain-agnostic functionality considers all values in each entity profile rather than relying on a particular (set of) attribute(s).

JedAI has been presented as a demo to two different communities, namely Semantic Web [27] and databases [28]. In this work, we present its structure and characteristics in more detail, introduce the new features of version 2.1, and provide experimental evidence of its performance over real data.

The rest of the paper is structured as follows: Section 2 delves into JedAI's architecture, Section 3 elaborates on the new features in version 2.1, Section 4 presents experiments that highlight the potential of JedAI, and Section 5 concludes the paper along with directions for future work.

2. ARCHITECTURE

JedAI's architecture appears in Figure 2. It is *modular* so that it can be easily extended by expert users in the future. Every component implements a simple (Java) interface such that every new class (algorithm) implementing it can be seamlessly added.

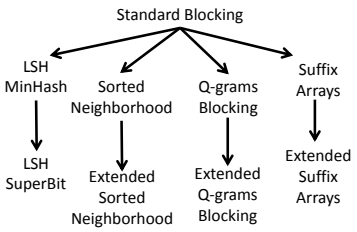

Figure 3: The Block Building methods.

JedAI consists of two parts: *(i)* JedAI-core[7], the back-end that essentially constitutes a library of state-of-the-art methods, and *(ii)* JedAI-gui[8], the front-end that facilitates the use of the library. Below, we describe each part in detail.

2.1 JedAI-core

Figure 1 depicts the workflow implemented by JedAI-core, which consists of the following eight steps:

1) *Data Reading* loads from the disk into main memory the data collection(s) to be processed along with the respective golden standard. The following data formats are supported: CSV, XML, OWL and RDF files as well as relational databases and SPARQL endpoints. Any mixture of these formats is possible in case of Clean-Clean ER. This is made feasible by transparently converting entities of any format into a flat name-value pairs model.

2) *Schema Clustering* is an optional step that groups together syntactically similar attributes, which share similar names and/or values. Unlike Schema Matching, this step does not seek semantically identical attributes (e.g., "place" and "location"). Instead, it aims to improve the performance of the next steps by raising precision significantly with no impact on recall [28]. Three methods are currently supported, Attribute Value Clustering, Attribute Name Clustering, Attribute Holistic Clustering [23], but at most one of them can be added in a workflow. They can be combined with any technique offered by the Text Processing component (see Figure 2) for comparing aggregations of strings. All pairs of attributes with a similarity above a% of the maximum similarity for either attribute are placed into the same cluster [29].

3) *Block Building* clusters similar entities into blocks so as to drastically reduce the candidate match space and to cut down on the running time. JedAI includes the nine methods in Figure 3, where every edge $A \rightarrow B$ denotes that method B is built on top of A, using the same core structures in a different way. For more details on the internal functionality of each method, please refer to [22]. All methods operate in a schema-agnostic fashion, extracting several signatures from every entity to place it into multiple blocks. The resulting redundancy yields high recall at the cost of low precision [4, 22].

[7]Code available under Apache License V2.0 at: `https://github.com/scify/JedAIToolkit`

[8]Code available under Apache License V2.0 at `https://github.com/scify/jedai-ui`.

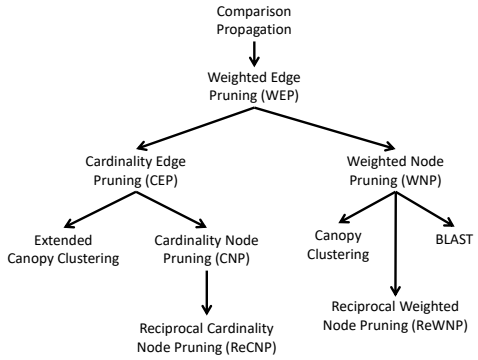

Figure 4: Comparison Cleaning methods.

4) *Block Cleaning* is an optional step that improves time efficiency by cleaning the original set of *overlapping* blocks from *redundant* and *superfluous* comparisons; the former are repeated across different blocks, while the latter involve non-matches [6, 22]. This step includes three complementary methods that operate at the level of entire blocks: Block Purging, Block Filtering, Block Clustering [8, 25].

5) *Comparison Cleaning* is another optional step that targets superfluous and redundant comparisons. Unlike Block Cleaning, though, it operates at the finer level of individual comparisons, offering a more accurate functionality at a higher computational cost. This step includes the ten methods that are depicted in Figure 4, where every edge $A \rightarrow B$ denotes that method B extends method A. Most of them are Meta-blocking techniques, described in [6, 26]. Only one of them can be selected, as they are competitive with each other.

6) *Entity Matching* conveys two schema-agnostic methods that carry out all comparisons in the final set of blocks: Group Linkage [21] and Profile Matcher, a custom approach that aggregates all attribute values of an entity into a common representation. Both methods can be combined with a large variety of graph and bag representation models and several associated similarity metrics [10] that are implemented by the Text Processing component (see Figure 2). This step yields a *similarity graph*, where every node corresponds to an entity and a weighted edge connects every pair of compared entities.

7) *Entity Clustering* partitions the nodes of the similarity graph into equivalence clusters such that every cluster contains all entity profiles corresponding to the same real-world object. For Dirty ER, this step implements the seven most efficient state-of-the-art methods evaluated in [13]. For Clean-

Clean ER, it offers the prevalent method in the literature, namely Unique Mapping Clustering [18].

8) *Evaluation* estimates the performance of the end result with respect to the golden standard that was specified in Step 1. To this end, it reports a series of measures for effectiveness and time efficiency. This step also includes *Data Writing*, which stores intermediate or end results into any of the supported data formats. In case a structured format is selected (CSV or relational database), the output retains the original entity ids. When selecting a semi-structured format (XML, RDF or SPARQL endpoint), the user has to specify the URI prefix, in case it is not available, i.e., when the original data were structured. To store the output to databases or SPARQL endpoints, the user should also provide the necessary credentials, if applicable, along with the table and the dataset namespace, respectively.

Regarding the use of optional steps, Schema Clustering is indispensable for heterogeneous data sources that involve a large number of noisy attributes. In these settings, it reduces the computational cost of Block Building and provides useful information for Comparison Cleaning and Entity Matching. Block Cleaning is indispensable for block collections that exhibit a Zipf distribution, where the larger a block size is, the less blocks correspond to it. Comparison Cleaning should be used in all cases, at least for eliminating all redundant comparisons at no cost in recall via Comparison Propagation. In case of redundancy-positive blocks, a Meta-blocking approach should be used to drastically reduce the computational cost. The exact method that should be selected for every optional step depends on the data at hand, as there is no clear winner among them.

2.2 JedAI-gui

This desktop application conveys a user-friendly wizard that allows for building ER workflows in a straightforward way, simply by selecting among the available methods per workflow step. A unique characteristic is that it offers three configuration options that are suitable for both expert and lay users:

1) The *default configuration* associates every available method with recommended parameter values that consistently achieve high performance, as verified through an extensive experimental study [26].

2) The *manual configuration* leverages the Documentation component of JedAI-core (see Figure 2), which enriches every method with a JSON file that provides information about its parameter configuration: the name and a short description of each parameter, the type of values it receives (e.g., an integer or real number), the range of acceptable values as well as its recommended default value. JedAI-gui presents this information to the user in the form of tooltips that pop-up in the configuration windows.

3) The *automatic configuration* currently accommodates two established approaches [1]: (i) *grid search* exhaustively applies to each parameter a set of reasonable settings that have been determined experimentally [26], or are typically used by experts in practice. (ii) *random search* iteratively tries arbitrary configurations that lay within the range of acceptable values for each parameter. Both approaches apply to individual methods and entire workflows. In the latter case, two operations are supported: (i) *holistic configuration*, where all parameters of all methods in a workflow are simultaneously optimized, and (ii) *step-by-step configuration*, where the parameters of each method are gradually optimized, independently of the others, following the workflow execution order.

To make the most of these options, JedAI-gui supports a *workbench* functionality. The evaluation window summarizes the performance of all experiments, enabling users to investigate the impact of parameter fine-tuning on the quality of results. This applies to all possible levels of granularity – from one or more parameters in a particular method to one or more methods in an entire ER workflow. The workbench functionality also facilitates the performance evaluation of the >10,000 different workflows that can be derived from the combination of the available methods per workflow step.

Another major characteristic of JedAI-gui is the *data exploration* functionality. After specifying the data to be processed, the user is able to go through the golden standard and the corresponding entity profiles, observing their properties as well as the level of noise and heterogeneity they contain. By the end of a workflow execution, the user can also examine the equivalence clusters that have been formed, assessing the quality of the results.

3. NEW FEATURES IN VERSION 2.1

The new JedAI version, 2.1, extends both JedAI-core and JedAI-gui. The latter is enriched with a hierarchically-structured benchmark screen, which is a crucial feature for the workbench functionality, as it allows users to review all aspects of performance per method. In this way, users can identify the weak link in an end-to-end workflow and assess whether a better parameter configuration is required or it should be substituted by another method.

Another important new feature is the command-line interface that has been added to JedAI-core. This allows developers to easily test changes made in the implementation of a method and to evalu-

ate new methods in the context of an end-to-end workflow. It also facilitates users to test JedAI on a server, exploiting much higher computational power than commodity hardware.

Additionally, we altered the way Block Building is handled by both the command-line interface and JedAI-gui. Instead of selecting a single method, they are now able to combine the results of multiple approaches. In this way, we satisfy a user requirement, which was articulated by businesses that applied JedAI to incomplete or noisy data. As an example, consider a customer database that abounds in profiles with missing or erroneous information. Applying a single block building technique, such as bigram blocking, yields a set of blocks with low levels of redundancy. This means that the block co-occurrence patterns for matching entities are scarce, downgrading the performance of Block and Comparison Cleaning methods, which are crucial for deriving high quality candidate matches [26]. To leverage their performance, we can apply them to the union of blocks formed by two or more block building methods (e.g., bigram and trigram blocking), which provides much denser co-occurrence patterns.

Another user requirement was to update JedAI's output. In version 2, it simply comprised equivalence clusters of matching entities, without providing any evidence for the degree of similarity. As a result, the end result of JedAI could not be refined by expert users or domain-specific applications that incorporate additional, contextual information. This is now resolved in version 2.1, as each pair of matching entities is associated with a confidence score that indicates their profile similarity.

Finally, several new methods have been added in JedAI-core, such as Correlation Clustering and chi-squared weighting scheme for Comparison Cleaning. We also integrated the results of Schema Clustering into Comparison Cleaning and Entity Matching. As indicated in [29], evidence from attribute clusters, such as entropy, enhances significantly the weights assigned to entity pairs in the sense that it facilitates the distinction between matching and non-matching ones. The same principle has been incorporated into Entity Matching, weighting the contribution of n-grams to the overall pair similarity according to the corresponding attribute clusters.

4. EXPERIMENTS

We now evaluate the performance of JedAI with respect to the state-of-the-art in the literature. To this end, we use the four real-world, structured, Clean-Clean ER datasets that were introduced in [17]. Their technical characteristics are listed in Table 1. Note that D_1 and D_2 entail product data, while D_3 and D_4 involve bibliographic data.

Dataset	D_1	D_2	D_3	D_4
$Source_1$	Abt	Amazon	DBLP	DBLP
$Source_2$	Buy	Google Pr.	ACM	Scholar
$Entities_1$	1,076	1,354	2,616	2,516
$Entities_2$	1,076	3,039	2,294	61,353
NVP_1	2,568	5,302	10,464	10,064
NVP_2	2,308	9,110	9,162	198,001
Duplicates	1,076	1,104	2,224	2,308
Cartesian Pr.	$1.16{\cdot}10^6$	$4.11{\cdot}10^6$	$6.00{\cdot}10^6$	$1.54{\cdot}10^8$

Table 1: Dataset technical characteristics. NVP stands for attribute name-value pairs.

We applied the following workflow to all of them: Token Blocking for Block Building, Block Purging with size constraints along with Block Filtering for Block Cleaning, Cardinality Node Pruning (CNP) for Comparison Cleaning, Profile Matcher for Entity Matching and Unique Mapping Clustering (UMC) for Entity Clustering. This workflow involves five parameters: (i) the maximum block size (Block Purging), (ii) the ratio of retained blocks (Block Filtering), (iii) the weighting scheme (CNP), (iv) the representation model in combination with the similarity metric (Profile Matcher), and (v) the minimum similarity for a pair of matches (UMC).

To explore the potential of this workflow, we fine-tuned these parameters in three ways: (i) step-by-step random configuration, where we used the methodology of [26] for independently optimizing each method until CNP[9] and the F-Measure for optimizing the last two methods, (ii) holistic random configuration, whose goal is to maximize the overall F-Measure, and (iii) step-by-step grid configuration, where we used the same criteria as the first case. We compare these configurations against three state-of-the-art domain-specific, learning-based systems: (i) COSY, which is a commercial system[10] that achieves the top performance in [17], (ii) DeepMatcher [19], and (iii) Magellan [16]. For the last two, we consider the top performance that is reported in [19] among all configurations and dataset versions.

The F-Measure of all systems is reported in Figure 5(a). We observe that JedAI outperforms all baseline systems over D_1 by 15% to 45%, depending on its configuration. For D_3, the differences between all approaches are negligible (±1.5%), as they all achieve practically perfect performance. For D_2 and D_4, DeepMatcher achieves the top performance

[9]This methodology aims to maximize for each method the measure $PC(B){\cdot}RR(B,B')$, where B stands for the input blocks, B' for the output ones, $PC(B)$ for the pairs completeness, i.e., the recall of blocks B, and $RR(B,B')$=1-$\frac{||B'||}{||B||}$ for the reduction ratio - the decrease in pairwise comparisons when transforming B into B'.
[10]Due to license restrictions, its name is not disclosed.

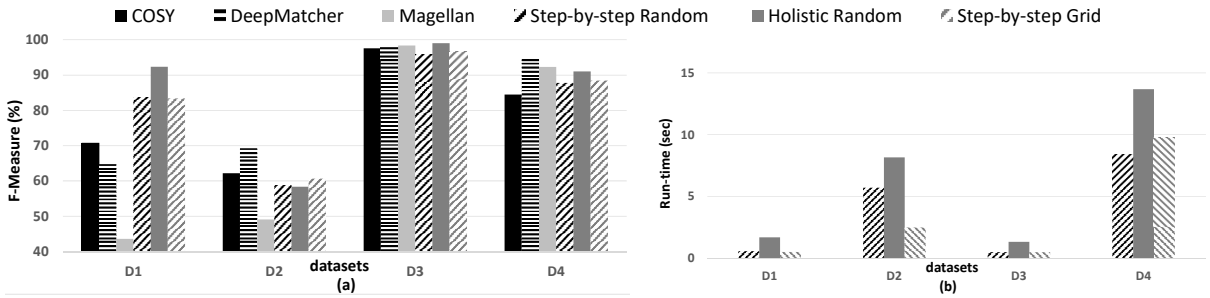

Figure 5: (a) Effectiveness of 3 different configurations of a JedAI workflow in comparison with 3 state-of-the-art domain-specific systems, and (b) the corresponding running times of JedAI.

to a significant extent, due to the external contextual information that is encapsulated in its features (i.e., word embeddings). JedAI is very close to COSY and outperforms Magellan to a significant extent over D_2, and vice versa over D_4.

It is worth associating these measurements with the corresponding time efficiency of each system. To measure JedAI's running time, we used a laptop with an Intel i7-4710MQ @ 2.50GHz, running Ubuntu 18.04.3 LTS and Java 8. We applied each configuration to every dataset[11], allocating 1 GB of RAM and performing 10 iterations with a clear Java cache. The average running times appear in Figure 5(b). We do not report the time performance of the baseline systems, as they all rely on manually-defined blocks of high performance, which are not reproducible, due to lack of details. We observe the high efficiency of JedAI's configurations, as they process every dataset within few seconds - less than 2 seconds for D_1 and D_3, from 3 to 8 seconds for D_2 and from 8 to 14 seconds for D_4. D_2 and D_4 are the most time consuming datasets, because they involve higher levels of noise and, thus, a larger number of candidate matches is processed. D_4 also involves the largest number of entities by far. Similar running times are reported in [17] for COSY, while Magellan and DeepMatcher require few seconds and few hours, respectively, for training their matching models, after having performed blocking with the help of an expert and labelling a considerable number of comparisons [19], operations that are very expensive in time. Therefore, we conclude that JedAI is much faster than the last two systems.

Another advantage of JedAI over Magellan is its ability to process Dirty ER datasets that Magellan cannot handle. This is illustrated in Table 2, which reports JedAI's performance over two established dirty datasets [3, 26]: Cora (1,295 entities, 17,184 pairs of duplicates) and CdDb (9,763 entities, 299 pairs of duplicates). We used the same system and approach for the time measurements and the same end-to-end workflow, except that UMC is replaced by Connected Components Clustering (recall that UMC does not apply to Dirty ER). We observe that JedAI achieves very high effectiveness, combined with very high time efficiency. The only exception is the high running time that is required for CdDb (between 2 and 4.5 minutes). This is caused by the large entity profiles, which involve 17.75 name-value pairs, on average (against 5.5 for Cora), and, thus, yield high levels of redundancy and a large number of candidate matches after CNP.

	Cora		CdDb	
	F-Measure	Run-time	F-Measure	Run-time
HRC	85.55%	514 msec	89.45%	266 sec
SRC	82.16%	294 msec	89.66%	155 sec
SGC	77.29%	3,752 msec	89.23%	127 sec

Table 2: Performance of JedAI over two Dirty ER datasets in combination with holistic random configuration (HRC) and step-by-step random and grid configuration (SRC and SGC, respectively).

Combined with the results in Figure 5, we observe a trade-off between the holistic and the step-by-step configurations. The former optimizes the parameters of all methods in an end-to-end workflow simultaneously: in every iteration, a new random, but valid value is assigned to each internal parameter and the iteration achieving the highest F-Measure is selected as optimal. As a result, holistic random configuration is crafted for identifying the global maximum, unlike the step-by-step configurations, which optimize every workflow step independently of the subsequent ones and, thus, are prone to confining themselves in local maxima. Yet, step-by-step configurations yield very low running times, because every method is fine-tuned to minimize its computational cost for the best possible effectiveness. In contrast, holistic random configuration consistently exhibits a significantly higher computational cost, since it exclusively considers the F-Measure in its optimization. As a result, its

[11]The corresponding code is available here: `https://github.com/scify/JedAIToolkit/tree/mavenizedVersion/jedai-core/src/test/java/org/scify/jedai/configuration/version2_1`.

blocking methods are configured to return a relatively high number of candidate matches.

On the whole, these results indicate that with proper configuration, JedAI produces learning-free, domain-agnostic workflows with an effectiveness that is comparable to, or better than, high-end, learning-based, domain-specific ER solutions, while exhibiting very low running times and limited memory consumption across various domains.

5. CONCLUSIONS

We presented JedAI, a user-friendly ER toolkit that fulfills the two main challenges arising in data integration [12]: the development of extensible, open-source tools, and the provision of solutions that apply not only to structured, but also to semi- or even unstructured data. We described JedAI's unique characteristics and elaborated on its main components, highlighting the new features in version 2.1. We demonstrated the high performance of its workflows and verified that it is ideal for the development phase of ER solutions, as it facilitates the identification of the best end-to-end workflows for a particular dataset and use case. Yet, the resulting workflow should be optimized for production systems.

In the future, JedAI will support pre-trained embeddings and will also allow for combining multiple algorithms and/or representations models for the pairwise comparisons during Entity Matching. Special care will be taken to progressively evaluate the performance of end-to-end workflows, a feature that is particularly useful when processing very large datasets. In later versions, we intend to enrich JedAI with support for supervised learning techniques. To comply with its domain-agnostic foundations, we will begin with approaches that leverage generic, schema-agnostic features, like those in [2]. Upon successful completion of this extension, we will also consider schema-based features and constraints, which call for fundamental changes for their incorporation. Special care will be taken to add active learning techniques from top crowd-sourced ER approaches [7, 9], thus addressing the sensitivity to cluster size for some of JedAI's workflows.

Acknowledgements. This work was partially funded by the EU project ExtremeEarth (825258).

References

[1] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *JMLR*, 13:281–305, 2012.
[2] G. D. Bianco, M. A. Gonçalves, and D. Duarte. BLOSS: effective meta-blocking with almost no effort. *Inf. Syst.*, 75:75–89, 2018.
[3] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Trans. Knowl. Data Eng.*, 24(9):1537–1555, 2012.
[4] V. Christophides, V. Efthymiou, and K. Stefanidis. *Entity Resolution in the Web of Data*. Morgan & Claypool Publishers, 2015.
[5] S. Das, P. S. G. C., A. Doan, J. F. Naughton, G. Krishnan, R. Deep, E. Arcaute, V. Raghavendra, and Y. Park. Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services. In *SIGMOD*, pages 1431–1446, 2017.
[6] X. L. Dong and D. Srivastava. *Big Data Integration*. Morgan & Claypool Publishers, 2015.
[7] D. Firmani, B. Saha, and D. Srivastava. Online entity resolution using an oracle. *PVLDB*, 9(5):384–395, 2016.
[8] J. Fisher, P. Christen, Q. Wang, and E. Rahm. A clustering-based framework to control block sizes for entity resolution. In *KDD*, pages 279–288, 2015.
[9] S. Galhotra, D. Firmani, B. Saha, and D. Srivastava. Robust entity resolution using random graphs. In *SIGMOD*, pages 3–18, 2018.
[10] G. Giannakopoulos, P. Mavridi, G. Paliouras, G. Papadakis, and K. Tserpes. Representation models for text classification: a comparative analysis over three web document types. In *WIMS*, pages 13:1–13:12, 2012.
[11] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. W. Shavlik, and X. Zhu. Corleone: hands-off crowdsourcing for entity matching. In *SIGMOD*, pages 601–612, 2014.
[12] B. Golshan, A. Y. Halevy, G. A. Mihaila, and W. Tan. Data integration: After the teenage years. In *ACM PODS*, pages 101–106, 2017.
[13] O. Hassanzadeh, F. Chiang, R. J. Miller, and H. C. Lee. Framework for evaluating clustering algorithms in duplicate detection. *PVLDB*, 2(1):1282–1293, 2009.
[14] R. Isele and C. Bizer. Learning expressive linkage rules using genetic programming. *PVLDB*, 5(11):1638–1649, 2012.
[15] P. Konda, S. Das, P. S. G. C., A. Doan, A. Ardalan, J. R. Ballard, H. Li, F. Panahi, H. Zhang, J. F. Naughton, S. Prasad, G. Krishnan, R. Deep, and V. Raghavendra. Magellan: Toward building entity matching management systems. *PVLDB*, 9(12):1197–1208, 2016.
[16] P. Konda and S. D. et. al. Technical perspective: : Toward building entity matching management systems. *SIGMOD Record*, 47(1):33–40, 2018.
[17] H. Köpcke, A. Thor, and E. Rahm. Evaluation of entity resolution approaches on real-world match problems. *PVLDB*, 3(1):484–493, 2010.
[18] S. Lacoste-Julien, K. Palla, A. Davies, G. Kasneci, T. Graepel, and Z. Ghahramani. Sigma: simple greedy matching for aligning large knowledge bases. In *KDD*, 2013.
[19] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra. Deep learning for entity matching: A design space exploration. In *SIGMOD*, pages 19–34, 2018.
[20] M. Nentwig, M. Hartung, A. Ngomo, and E. Rahm. A survey of current link discovery frameworks. *Semantic Web*, 8(3):419–436, 2017.
[21] B. On, N. Koudas, D. Lee, and D. Srivastava. Group linkage. In *ICDE*, pages 496–505, 2007.
[22] G. Papadakis, G. Alexiou, G. Papastefanatos, and G. Koutrika. Schema-agnostic vs schema-based configurations for blocking methods on homogeneous data. *PVLDB*, 9(4):312–323, 2015.
[23] G. Papadakis, E. Ioannou, T. Palpanas, C. Niederée, and W. Nejdl. A blocking framework for entity resolution in highly heterogeneous information spaces. *IEEE Trans. Knowl. Data Eng.*, 25(12):2665–2682, 2013.
[24] G. Papadakis and W. Nejdl. Efficient entity resolution methods for heterogeneous information spaces. In *ICDE Workshops*, pages 304–307, 2011.
[25] G. Papadakis, G. Papastefanatos, T. Palpanas, and M. Koubarakis. Scaling entity resolution to large, heterogeneous data with enhanced meta-blocking. In *EDBT*, pages 221–232, 2016.
[26] G. Papadakis, J. Svirsky, A. Gal, and T. Palpanas. Comparative analysis of approximate blocking techniques for entity resolution. *PVLDB*, 9(9):684–695, 2016.
[27] G. Papadakis, L. Tsekouras, E. Thanos, G. Giannakopoulos, T. Palpanas, and M. Koubarakis. Jedai: The force behind entity resolution. In *ESWC*, pages 161–166, 2017.
[28] G. Papadakis, L. Tsekouras, E. Thanos, G. Giannakopoulos, T. Palpanas, and M. Koubarakis. The return of jedai: End-to-end entity resolution for structured and semi-structured data. *PVLDB*, 11(12):1950–1953, 2018.
[29] G. Simonini, S. Bergamaschi, and H. V. Jagadish. BLAST: a loosely schema-aware meta-blocking approach for entity resolution. *PVLDB*, 9(12):1173–1184, 2016.

Natassa Ailamaki Speaks Out on How to be a Systems Researcher and How to Manage a Large Research Group

Marianne Winslett and Vanessa Braganholo

Anastasia Ailamaki

https://people.epfl.ch/anastasia.ailamaki?lang=en

Welcome to ACM SIGMOD Record's 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 Anastasia Ailamaki, who's a professor at the Swiss Federal Institute of Technology, better known as EPFL. Before that, Natassa was a professor at Carnegie Mellon. She's an ACM Fellow, a Sloan Fellow, and received the European Young Investigator Award, as well as ten Best Paper awards. After this interview, she received the Edgar F. Codd Innovation Award from the ACM SIGMOD in 2019, and the Nemitsas Prize in Computer Science from the President of the Republic of Cyprus in 2018. Her Ph.D. is from the University of Wisconsin, Madison. So, Natassa, welcome!

You work in two very different areas, new hardware and information management for the life sciences. Why those two?

It started that way. I started as a student. I worked with Yannis Ioannidis initially and then with David DeWitt when Yannis moved back to Greece. The work that we were doing with Yannis involved interaction with the Soil Science group, at the time, in Wisconsin. We were building a data management infrastructure for them. There were very interesting challenges to meet. I liked the topic, but then I ended up doing a Ph.D. on architecture-conscious data management. I also discovered my love for interdisciplinary science through these two projects.

Afterwards, as an assistant professor, I always thought that I should stop one of the two thrusts, but then it got more and more interesting on both sides. On the architecture side, we got a lot of interesting devices. Hardware trends, as you know, go upward and onward in many directions. There were (and still are) many problems to solve, and I had a lot of students interested. And on the other side, I also started working with astronomers, with mechanical engineers, earthquake simulation people. Then, when I moved to EPFL, I started talking with life scientists, neuroscientists in particular, clinical and experimental. The interesting problems just flourished. So, I ended up working on both sides. It's actually been very motivating to me.

In your research on new hardware, how can you be sure that your insights into hardware-related effects aren't an artifact of running student-quality software?

That's a very good question. The quality of student software varies wildly, and one cannot be fully sure that it's thoroughly tested. I get away with less work testing it very closely on the experimental side of my work: the numbers have to be generated and tested and re-generated and corroborated in so many different ways that the software that we write is mostly scripts to test what's going on and figure out whether, for example, the breakdowns of time reveal any bottlenecks and where in the machine the time is attributed to and things like that. It's not production software, it's not something intended for end users. It's meant to reveal how the software uses the underlying hardware, and that we can test in many other ways than just looking at the result and making sure it's correct.

On the other hand, when we're doing work for scientific databases for actual users, there we have more problems. There, a lot of times, we need the involvement of an engineer to make sure that the software is tested correctly, does the right thing for the user, is reliable and so on and so forth.

How have the changes in the hardware world affected relational databases up until now?

The hardware world has been giving us quite a few opportunities, taking away decision-making processes at the hardware level, and giving us, essentially, choice. The big example there is parallelism. We view its different facets as they come up: instruction-level parallelism, pipelining, now multi-core, multi-threading, and so on and so forth. But it's been pretty much the same, explicit or implicit parallelism methods revealing opportunities at the hardware level.

Now, databases, from the very beginning of time, have been called to use this parallelism that's available in the hardware. We are, more or less, trying to go towards the same general direction. However, it's not always the same because the details of how the parallelism is implemented, implicitly or explicitly, are important for the architecture of the system design that we will end up with for the final product.

The other important aspect, however, that's come up very recently is heterogeneity in the hardware, heterogeneity at the compute level, at the interconnect level, and at the memory level, more recently. There, we're really not ready to use what's available. I foresee a very interesting future in hardware/software co-design for databases, just because this is the first time, really, we get something fundamentally new.

Hardware/software co-design means that you design software while keeping in mind what the hardware can do. And you also, at the same time, have the hardware get influenced by the software when you make design decisions at that level.

Tell me more about that co-design.

Well, when you architect a database engine, you write code for it. When someone tells you what's going on with the hardware, the first reaction is "why should I care?" Right? Because there is an operating system in

between. There is other software and lots of stuff happening. You're writing middleware; databases are middleware. They're going to run other applications, but they're not really that close to the hardware. People will go as far as data placement, and then they will stop there. They won't automatically think that they have to really consider what's going on in the compute subsystem or the high-level memory subsystem.

And they should!

Well, you and I may think that they should. And I certainly didn't think so until I realized that they should, after reading and experimenting and realizing that the operating system actually does very, very little and for a very small part of the time. There's really nothing between the database system and the hardware that impedes very good communication between the two. But this communication can only be good, which means the database system really taking advantage of the hardware resources, if the database system actually is aware of the hardware resources. So, co-design means that you design both of the things at the same time. Hardware/software co-design means that you design software while keeping in mind what the hardware can do. And you also, at the same time, have the hardware get influenced by the software when you make design decisions at that level.

But from our point of view, what we really need to do is to, first of all, evaluate the current situation because otherwise, you will run into a chicken and egg problem, a vicious cycle: which is designed for what… We want database software to be sort of general-purpose. We don't want to completely and deeply vertically-integrate everything. But at the same time, we also want the hardware to be able to run a vast breadth of applications and not just database systems. So, there are interesting tradeoffs to be solved.

I find very rewarding to work with micro-benchmarks: look at really simple fundamental database operations, and figure out, when they are executed (and they're executed very often), how they use the hardware resources. I've been doing this with many generations of databases and many generations of several different brands of computers.

I need an example of a couple of those fundamental database operations.

Like, for example, an index probe or even sorting for larger scale or a join. These are the kinds of operations that you can look at. Now, why are these important at the micro-architecture level?

Well, you may be executing a very short transaction where an index probe is the main action, or you may be executing two-megabyte-long code for one query where the index probe is a millionth of what it does. But in both cases, the processor grabs a bunch of instructions, gets them through the pipeline, graduates them, executes them, gives out the results, then grabs the next bunch of instructions. So the index probe code is as much as the window "seen" by the pipeline of execution at the processor level.

What happens is there's this window of instructions that moves; that window is very, very small. By basically measuring what's going on with those micro-benchmarks, I found it pretty educating to understand how the interaction between the software and the hardware works, how smoothly it works, whether the processor was sitting there idle or whether it was actually doing useful work.

So, co-design, to go back to your previous question, means that we work out details in the architecture of our engine keeping in mind how the hardware receives both the instruction stream of the code that we write, but also how it will be able to feed the high-level caches with data in the memory subsystem, without having to thrash them, and how we can also, ourselves, sort and schedule those accesses today, the instructions, so that we can achieve a better result.

The phrase co-design sounded exciting to me because it sounds like we were gonna get some control over what the hardware was gonna be like.

I had the same impression and the same hope. It actually isn't completely hopeless. There are a lot of times where we come up with an intuition for an algorithm, and we test the algorithm. That algorithm works really well, but then there is a software part that could be done a lot more efficiently if the hardware could just implement a very inexpensive hint, for example, that could alert the software that an event has happened: the cache is full or something like that.

I have had meetings with people at hardware companies, at Intel, for example, where we had these discussions for a long time. There were actually very reliable results that we were presenting, that they were convinced that this would be a great thing. I don't know that any of these ever went to production because the time from when one person, no matter how high up the management hierarchy they are, realized that this could be a great thing and the time that this might make it to production overlap with different generations in the company roadmap. So, priorities change.

I have good news for you. Andrew Chien, just a few minutes ago, told us that we're gonna get our own SoCs (System on a Chip).

(jokingly) So, your own personal SoCs!

No. I mean personal to the database world. But I guess it's a long, slow process. You've probably been having those discussions a long time.

So, getting database-specific systems on-chip, I think it's a great idea. But this is just one of the things that will happen. I really believe we're heading to more–different futures that will all happen concurrently.

What are the other futures?

Database engines, transactional engines, will be different, depending on the hardware that they're executing on. This is a realization that comes from the necessity to move the line of abstraction, of independence, if you will, from the hardware, from the very low level that it's at right now to a bit higher, to obtain better utilization of the resources. It's sometimes simply impossible to do that, just because the resources at this point don't only give us more parallelism, but hardware also gives us a lot of heterogeneity. So, we get different kinds of cores on the same chip. And in order to use them, we have to really alter the architecture of the system.

Virtual and dynamic solutions only work up to a point. Then they impose so much housekeeping overhead, so long of a "case" statement, that you will not be able to execute them efficiently.

It sounds like it could be very painful for the database companies to handle all these different types of architectures unless they have a very clean way of...

It could be painful, but I view it more as an opportunity. First of all, the research community is going to be given a lot of new questions to answer. Second, as I said, it's a matter of raising the abstraction line that divides the hardware-oblivious from the hardware-conscious part of the system. So, it really means that there will be multiple products coming out, which means growing the market and getting more people to really like databases. I see it really as an opportunity.

For me it's a fundamental change of scene because heterogeneity has never been this accessible from us. I have been doing work with CPUs and GPUs or CPUs and NPUs for research long time ago, but the other processors that weren't the main CPU were treated like the poor cousin that we would enlist just to see what happens, get a result, answer an academic question we had posed ourselves but nobody really cared about in the industrial world. Now, this changes a lot. I'm very excited.

It's exciting times!

What changes do you see on the horizon in the kinds of applications that use relational databases? I mean, new applications.

Changes in the form of an application being one way now and changing in the future? I'm not sure I see any changes whatsoever. What I see, however, is new applications that come up, given that we are evolving, in so many ways, the infrastructure that we build that more and more things are possible. Now, for example, more and more systems involve just-in-time access to data. So, NoETL is a big wave. My startup is in that domain. We access data that just arrived in multiple formats, from multiple sources. We don't access it unless it's asked for, unless it's queried. That's a big change.

Applications are not ready for something like that. But from the moment that this sinks in, for example, there will be a lot more interesting opportunities at the application level, to take advantage of this stuff. Apps are too bound, now, to the ingest-then-query model.

What do you think of Spark Streaming and Storm as ways to handle that data?

They're great enablers of handling data on a distributed infrastructure. They free the user from having to worry about all of these things, like Hadoop configuration and all that nitpicking that one has to go through in order to make things work today at scale.

But since you have your startup, you must not feel that they solve all the problems that matter for these new applications.

As I said, they're enablers. They're mechanisms. It's up to the application or the middleware infrastructure to solve the problem. It's a tool. You have to use the tool correctly to solve the problem. It's not a policymaker in the software.

How are we doing in the database world with respect to energy efficiency?

We have been doing pretty well in investigating what we can do. I don't think we've hit home run yet in understanding the relationship between database management systems and energy efficiency because, in

a lot of ways, it's a moving target. We have industry that is very qualified to actually do whatever they can to drop the numbers in the power bills. We have research people at all levels, at the architectural level, even lower, and then on top of that, at the system level, systems researchers, and even database researchers that do work in the area. But all of this work is coming out at the same time. We're taking each other's work and trying to measure what's happening in our world.

I don't think we've hit home run yet in understanding the relationship between database management systems and energy efficiency [...]

It's an interesting and very dynamic research environment right now. We haven't really hit a home run yet, I don't think, even in understanding the problem rather than solving it. Because the problem is very close to numbers and quantifying such a problem requires a very big breadth of numbers that are constantly changing today.

But I think that we will have to solve this problem. Eventually, the rest of the factors will converge. We will know that unless we do something in the software that's fundamentally energy-conscious, these numbers are not going to go down, the costs are not going to go down. So, we will have to solve a more stable problem, hopefully, in the near future. It's certainly a very interesting area.

You are already viewed as a rock star researcher in Europe. I've been told, this week, that you are much better than you think you are in every way, which suggests that your fame is only going to increase. How do you deal with the demands that come with this?

I'm very happy to be here. I take demands as they come. Whenever I receive a request to do some work or give a talk, I evaluate it honestly. If I can do it, I do it. If I can't, I just can't. The biggest problem that all of us face at some point, very early on, even, is to be able to say "no". Right? I am very, very bad at that. I don't say as many "no"'s as I should and a lot of times it takes a toll on my other work, that I have accepted too much to do on the outside. But at the same time, I find it very rewarding. Even for things that I regret immediately having said "yes" to, then I realize that there was a very positive side to this after I've done the work. So, I've never really regretted doing something even if it meant that it was a lot of work. But I am trying to train myself to say a bit more "no"'s. We should all try to do that, I think.

Your research group is fairly large and you're known as being hands-on with your students. How do you scale up being hands-on to a group of a dozen people or so?

I have twelve students, ten to twelve. The number varies in that neighborhood. Then I have a few engineers because now I'm in Europe and European Union projects demand engineering work. I have post-docs and scientific collaborators, who come after their Ph.D. Then, I have interns and masters students. It's a healthy group of 20 to 25 people. So, I organize meetings with everybody on a weekly or bi-weekly basis.

Individual?

Individually, depending on demands and depending on my travel as well. So I end up seeing everyone at least every two weeks. As is normal, I need to see mostly the graduate students. They take priority. Then I see the post-docs and the engineers as needed.

I am proud to say that this is a great group. The people who apply and are admitted to EPFL, and the people who I receive and end up in my group, they are top people. My students are fantastic. Everybody, post-docs, engineers, my administrative staff, is great. EPFL is a very supportive place. They have a great infrastructure, lots of resources. So, I find that my day-to-day job there, at least from that point of view, is very, very easy. I don't see any trouble.

The male professors want me to ask you how they can find more female Ph.D. students.

I don't particularly try to get female Ph.D. students. I always hire on a merit basis. One realization that I had when I went to Europe is that intelligence is uniformly distributed. I was worried that I wasn't going to have as good students as I had at CMU, just because I was at EPFL. It turns out that I am getting just as good students. But maybe male professors should realize that intelligence is also uniformly distributed not only geographically but also gender-wise. Giving equal opportunities doesn't come automatically. Perhaps we need to work toward that direction.

That's interesting because I did not find that to be the case at Illinois. I was the head of the awards committee, and the awards winners were very disproportionately female (considering how few

women we had). So, I realized that, in fact, our average female was smarter, significantly.

I believe that, but I would believe the same if you told me that it was male. I think you are describing a snapshot. We should just forget about gender. That has been – if I have one big change that I wish it could just magically happen is that, upon entrance into my building of work, I could be a genderless person. My life would be so much easier. I think that a lot of people would agree with me, from both genders, that that would be a very, very good idea. However, human nature doesn't allow that. So, we have to work with what we have.

But as far as students are concerned, the one thing to remember – talking about working with what we have and getting a tight grasp of reality here – is that female graduate students, on average, are way less confident than male graduate students. That's something that comes from nature (it is the big word). I don't know how to specialize it more. But I want to work with the result. I've always learned to look at the result of my experiment and reverse engineer, not try to find the source. Unless the problem is solvable, I'm not interested, and I think that this problem is not solvable at the source. We need to treat the symptoms.

So, I receive this graduate student, who is way less confident than she should be. And I take it upon myself to actually work with her: channel her motivation the right way; make her understand what she's doing well, what she doesn't do well; and make sure that all her capabilities, at the end of her tenure of five years, whatever, in my group, reach the level that will allow her to have a good job and a good life.

But wait a minute. That's exactly what I'm trying to do with the male students, as well. It's just that some people, male or female, are better at some things and they're not as good at some other things. As professors, basically, our job is to have this person across from us once a week (or more often), and to try to help them surface their best qualities and apply them to whatever it is they are passionate about.

I like that summary. How do you get so much energy?

(jokingly) You mean like the energy efficiency we were talking about?

No! No! No! No!

I understand. I think I'm naturally pretty energetic as a person. I don't know how I am like that. But I have a very supportive environment. Not just at work. I described that already, and I couldn't have wished for a better environment. The first half of my career at CMU, the second half of my career at EPFL; both environments have been the most nurturing I could have ever hoped for. I cannot emphasize this enough.

But also, my personal life. I have my ups and downs, like everybody. I've had my troubles, like everybody. But every time in my life there's been a lot of good stuff happening. So, I try to be appreciative, even when things are not the best of what's going on, and start every day as a new opportunity. I know that sounds as a cliché, but we do have to look around and see that there's great, great things going on. And not just at work. Our families, our parents, our children, where we live, what we can do during one single day… There's a lot of good stuff.

That's a lovely answer!

[…] the beauty of computer science is that the size and time change the solutions to the same problem. We can actually innovate in different ways solving similar problems as our predecessors, but really pushing the envelope of science, and of technology of course, much further.

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

I don't know that I have so much advice, because advice is something that you're very sure is going to work for the other person and that's a very big crowd. But actually, what worked for me was to try to look back to the research that people did when things were relatively simple at the infrastructure level. I can only know how to be a systems researcher.

With systems, history is a very important education factor. I found a lot of inspiration in the work that the first database groups did. And I still do. I find myself going back and looking at it very often; more often than I read blogs. That's one thing: drawing inspiration from the mental ideas because the beauty of computer science is that the size and time change the solutions to the same problem. We can actually innovate in different ways solving similar problems as our

predecessors, but really pushing the envelope of science, and of technology of course, much further.

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

Two things. One would be I would give a lot more talks to junior students, both undergraduate and graduate. Not necessarily computer science talks but general talks about what I feel a good technical talk should be structured like. Stuff that colleagues of mine do, as well, and I'm very grateful that they do, but I would like to also do some more service in that direction.

Then at large, not within what I do right now, I would like to get some education and perhaps even a degree in a different science. I've always been fascinated by architecture and material sciences, so I would really go toward that direction to get, maybe, a degree or definitely some classes there.

Do you mean computer architecture or building architecture?

No, building architecture.

Civil engineering. Cool.

Thank you very much for talking with us today.

Thank you very much. It was a pleasure.

The Seattle Report on Database Research

Daniel Abadi, Anastasia Ailamaki, David Andersen, Peter Bailis, Magdalena Balazinska, Philip Bernstein, Peter Boncz, Surajit Chaudhuri, Alvin Cheung, AnHai Doan, Luna Dong, Michael J. Franklin, Juliana Freire, Alon Halevy, Joseph M. Hellerstein, Stratos Idreos, Donald Kossmann, Tim Kraska, Sailesh Krishnamurthy, Volker Markl, Sergey Melnik, Tova Milo, C. Mohan, Thomas Neumann, Beng Chin Ooi, Fatma Ozcan, Jignesh Patel, Andrew Pavlo, Raluca Popa, Raghu Ramakrishnan, Christopher Ré, Michael Stonebraker and Dan Suciu

ABSTRACT

Approximately every five years, a group of database researchers meet to do a self-assessment of our community, including reflections on our impact on the industry as well as challenges facing our research community. This report summarizes the discussion and conclusions of the 9th such meeting, held during October 9-10, 2018 in Seattle.

1. INTRODUCTION

From the inception of the field, academic database research has strongly influenced the state of the database industry and vice versa. The database community, both research and industry, has grown substantially over the years. The relational database market alone has revenue upwards of $50B. On the academic front, database researchers continue to be recognized with significant awards. With Michael Stonebraker's Turing Award in 2014, the community can now boast of four Turing Awards and three ACM Systems Software Awards.

The strong progress the database research community has made in recent years is clearly evident. Over the last decade, our research community pioneered the use of columnar storage, which is used in all commercial data analytic platforms, whether or not they are based on a relational engine. Database systems offered as cloud services are widely used and have witnessed explosive growth. Hybrid transactional/analytical processing (HTAP) systems are now an important segment of the industry. All data platforms have embraced SQL-style APIs as the predominant way to query and retrieve data. A new generation of data cleaning and data wrangling technology is being explored. Database researchers have played an important part in influencing the evolution of streaming data platforms as well as NoSQL systems.

Our achievements show that the state of our community is strong. Yet, in technology, the only constant is change. Today, we are living in a data-driven society where decisions are increasingly driven by the insights gathered from analysis of relevant data ("data is the new oil"). This societal transformation places us squarely in the center of technology disruptions. However, the fact that data is at the center of everything today also means that the field has grown in breadth and that new challenges have arisen. Indeed, just in the last five years, much has changed in industry and the research community. Technology trends are providing our community with an unprecedented opportunity to have an even bigger impact in today's data-driven world than ever before.

In the Fall of 2018, the authors of this report met in Seattle to identify and discuss research directions for the community that seem especially promising, considering the key developments that impact our field. There is a long tradition of such meetings in our community, held approximately every five years. The first such meeting was held in conjunction with VLDB 1988 [3] and the last one prior to Seattle took place in Irvine in 2013 [2].

This report summarizes the findings from the Seattle meeting of database researchers. We begin by reviewing key technology trends that impact our field. The central part of the report covers research themes that we believe are especially promising. We close by discussing steps the community can take to be more impactful beyond solving the technical research challenges.

2. WHAT HAS CHANGED IN THE LAST FIVE YEARS?

The transformation to data-driven decision making has been in progress for many years. In fact, the last report identified Big Data as our field's central challenge [2]. However, in the last five years, the transformation has accelerated well beyond our projections, in part due to technological breakthroughs in machine learning (ML) and artificial intelligence (AI). Deep neural networks (DNNs) have led to unprecedented progress in image analysis and natural language processing (NLP), among other disciplines. Reinforcement learning emerged as a powerful paradigm to complement traditional supervised learning. Recently, models such as BERT hold the promise of democratizing the use of natural language as an interaction model for tasks in any enterprise, not just Internet companies with a rich information corpus. The barrier to writing ML-based applications has been sharply lowered by widely available programming frameworks, such as TensorFlow and PyTorch, as well as new FPGA, GPU, and specialized hardware for use in private and public clouds. The database community has a lot to offer to ML users given our expertise in data discovery, versioning, cleaning and integration. These technologies are critical for a machine learning platform to derive insights from data. Execution of inference and training workflows can potentially benefit from query optimization techniques. The database community can also help shape how traditional SQL querying functionality is seamlessly integrated with machine learning. Moreover, with the increasing availability of usage data, ML can be leveraged to transform the database platform itself.

A related development has been the rise of data science as a discipline that combines elements of data cleaning, transformation, statistical analysis, data visualization, and machine learning techniques. Today's world of data science is quite different from the previous world of statistical tools such as SAS and SPSS, and from traditional data transformation tools in the enterprise data integration world. Notebooks have become by far the most popular interactive environment. Our expertise in declarative query languages can enrich the world of data science by making it more accessible to domain experts, especially those without traditional computer science background.

Our society has become increasingly concerned about the state of data governance. This is a difficult challenge as data moves within information systems as well as across organizational entities and national borders. Data governance requires that data owners adhere to data privacy and other constraints related to the movement of data. To meet this requirement, data provenance and metadata management technology are important ingredients. Data governance has also led to the rise of confidential cloud computing whose goal is to exploit cloud resources while keeping the data encrypted. Beyond data governance, another societal concern is ethical and fair use of data. This concern impacts all fields of computer science, but is especially important for data management, which must enforce such policies.

The last report observed that "Cloud Computing has become mainstream" [2], and indeed, usage of managed cloud data systems has grown tremendously in the last five years. As an alternative to provisioned resources, the industry now offers on-demand resources that provide extremely flexible elasticity, popularly referred to as *serverless*. For analytics, the industry has converged on a *data lake architecture*, which uses elastic compute services to analyze data in cloud storage "on-demand". The elastic compute could be jobs on a Big Data system such as Apache Spark, a traditional SQL data warehousing query engine, or an ML workflow. It operates on cloud storage with the network in-between. This architecture disaggregates compute and storage, so they can scale independently. These changes have profound implications on how we design future data systems.

Industrial Internet-of-Things (IoT), focusing on domains such as manufacturing, retail, and healthcare greatly accelerated in the last five years, aided by versatile connectivity, cloud data services, and data analytics infrastructure. Its requirements have further stress-tested our ability to do fast data ingestion and quickly discover insights with minimal delay for real-time scenarios such as monitoring. Their effectiveness also depends on efficient data processing at the edge, including data filtering, sampling, and aggregation.

Finally, there are significant changes in the hardware landscape. With the end of Dennard scaling, and the rise of compute-intensive workloads such as DNN, a new generation of powerful accelerators leveraging FPGAs, GPUs, and ASICs are being used. These technologies appear to be the only viable approaches today to train big models. The memory hierarchy continues to evolve with the ad-

vent of a new generation of SSD and low-latency NVRAM. Specialized interconnects as well as improvements in network bandwidth and latency continue to be remarkable. Beyond datacenters, the advent of 5G with ample bandwidth can reshape the workload characteristics of data platforms. These developments point to the need to leverage an increasingly heterogeneous hardware landscape as we rethink the architecture of the next generation of database engines.

3. RESEARCH CHALLENGES

While we have made progress in some of the key challenges articulated in the last report [2], many of the difficult questions remain relevant today. The changes described in the previous section present us with new scenarios that deserve our consideration as well. This report combines these two sets of research challenges, organized into four subsections. The first subsection addresses challenges in data science where our community can play a major role. The second focuses on the emerging societal concerns of data governance. The last two cover the cloud data services and the closely related topic of database engines. It should be noted that some of the challenges cut across multiple themes, e.g., machine learning.

3.1 Data Science

The NSF CISE Advisory Council[1] defines data science as a field that focuses on "*the processes and systems that enable the extraction of knowledge or insights from data in various forms, either structured or unstructured*" Over the past decade, it has emerged as a major interdisciplinary field and it will become even more important in the future.

Data science is used to drive important decisions in companies and discoveries in science. It is used for one-off decision making as well as for tracking Key Performance Indicators (KPIs) over time. From a technical standpoint, data science is about the pipeline from raw input data through data integration and wrangling, to data analysis, data visualization, and finally insights.

Through the history of database systems, users have extracted insights from their databases. They have used complex SQL queries, online analytical processing (OLAP), data mining techniques, and statistical software suites. The modern data scientist works in a different environment. Jupyter Notebooks are the new de-facto standard, and data scientists rely on a rich ecosystem of open-source libraries for sophisticated analysis, including the latest ML techniques. They also work with data lakes that hold structured and unstructured datasets with varying levels of data quality – a significant departure from carefully curated data warehouses. Furthermore, data science is fundamentally a multidisciplinary field with deep integration with an application domain, whether in science or industry. These characteristics have created new requirements for the database community to address, as discussed below.

Data integration and wrangling: Data scientists repeatedly say that data integration and data wrangling is 80-90% of their challenge. These are problems the database community has worked on for decades. Thus, it can bring a solid understanding of the core challenges and known solutions. In the past, we have focused much of our efforts on solving "point problems", e.g., algorithms for specific challenges such as entity resolution. Instead, we need to devote more efforts on the end-to-end data-to-insights pipeline, including understanding systems that go all the way from raw data to an end-user's desired outcome, such as a visualization of the answer to the user's question, or a prediction by a machine learning model.

Data context and provenance: Data scientists need to understand the quality of the results that they are getting from a data to insights pipeline. In traditional database applications, query results are assumed to be correct, complete, and fresh. The data is trusted because it was created by the same entity that consumed it. In modern applications, correctness, completeness, freshness, and trust cannot be taken for granted. Consumers need to know the degree to which these properties of their data hold and to reason about their impact. This requires understanding the context of the incoming data and the processes working on it. This is a classic data provenance problem, which involves tracking, integrating, and analyzing this metadata. Beyond explaining results, data provenance also enables reproducibility, which is key to data science, although it is especially difficult when data has a limited retention policy. This is an area where we need to focus our efforts.

Data management in support of machine learning: Data science pipelines require machine learning, including the latest techniques such as deep learning. The database community needs to embrace

[1] https://www.nsf.gov/cise/ac-data-science-report/CISEACDataScienceReport1.19.17.pdf

this new type of workload. In addition to developing efficient methods for executing such workloads (see Section 3.4), it needs to investigate declarative programming paradigms to specify and optimize all stages of machine learning pipelines (data discovery, data preparation, and model building). The management of models and machine learning experiments is an area where our community can provide rich support, including but not limited to model versioning. Data provenance is also important in machine learning, as it can help identify differences between test data and training data that cause models to lose accuracy.

Fast Exploration: To support exploratory analyses by data scientists, systems must provide interactive response times over Big Data, since high latency reduces the rate at which users make observations, draw generalizations, and generate hypotheses. More research is needed for at scale visualization and interactive query processing. Research is also needed in developing methods that make creating and debugging complex data science pipelines easier than writing code in an imperative language like Python. This work also requires a change in the database community conference culture, promoting user studies that assess the impact of novel technology on data scientists.

Modern data analysis and management, including data science, continues to move to public clouds where the data for analysis is drawn from data lakes. This change has significant implications, as discussed later in this report. Big Data, Data Science, and AI have also led to more diverse applications that often do not fit well with the relational model. Data management researchers should work on determining the right data models, query operators, storage schemes, and optimizers for emerging data-intensive applications. This requires participation in building end to end systems and experimentation with user applications.

Since the database research community has worked on many facets of data-to-insights pipeline, we are well positioned to have a big impact. However, we must collaborate with other disciplines so that our technological contributions are recognized and adopted widely. The time has come for our community to develop a data science agenda that builds on our strengths, attracts broad participation, and helps shape this emerging field.

3.2 Data Governance

The data management, and indeed the entire computer science community, has become socially aware. As technology continues to impact society in more profound ways, the database community must focus more attention on the societal impact of the technology they develop.

Today, end-users generate data that becomes input to many data-intensive applications. Some of it is about people: our homes have become "smart", with sensors located in doorbells, thermostats, and other appliances; virtual assistants have entered our living rooms; medical records are digitized; and social media is publicly available and widely popular. Data-intensive applications that use these data sources raise not only technical challenges but also those of privacy and ownership. Data producers have an economic and personal interest that the data is used only in certain ways. For instance, they might have licensed the use of their personal health records for medical research, but not for military applications. The European Union's General Data Protection Regulation (GDPR), which has gained wide adoption beyond Europe, is directly related to the issues of data privacy and data use. These topics of data governance are discussed below.

Data use policy and data sharing: In industry, data science pipelines are often complex and different sub-teams work on different steps of the pipeline: one team prepares the data; another builds models on the data, and yet another team accesses the data and models through interactive dashboards. Additionally, data science teams leverage multiple heterogeneous data sources in data lakes. The database community needs to develop tools that support collaborations around data, including labeling, annotating, exchanging, securing, discovering, and capturing provenance of data. Such collaborations and sharing must adhere to fine-grained access control and auditing requirements to ensure data is used by the right people for the right purpose. Data provenance technology is needed to support auditing at scale so that checks for legitimate usage can be implemented. Finally, as data volumes grow, we need to better techniques to compress data, move data to cold storage, and choose data to discard.

Data Privacy: As we continue to aggregate data, balancing the need for data privacy with analytical usage of such data for decision support has emerged as a key challenge. Cryptographic techniques as well as differential privacy have emerged

as a foundation for much of the privacy work, including in our community. However, it is still unclear in what way differential privacy may be embedded in the database platforms effectively without restricting the query surface significantly. Collaborations across organizations are also subject to privacy constraints and require techniques such as Multi-Party Computation.

Ethical data science: Machine learning models can contribute to bias and discrimination. Activities around surfacing and countering those problems have gained traction in research and practice. The bias often comes from the input data itself. Sometimes, it comes from insufficiently representative data used to train models. Our community can apply expertise in data quality and data integration to help address this problem. Responsible data management has emerged as a new research direction where the data management community has much to contribute. A related challenge is identifying data designed to misinform, e.g., on social media platforms. Addressing the above challenge requires inferring intent, as well as collaboration with the NLP, computer vision, and other communities.

3.3 Cloud Services

The movement of workloads to the cloud has led to explosive growth for cloud database services, which in turn has led to substantial innovation, experimentation, as well as new research challenges.

Challenges of new consumption models: The simplest consumption model is Infrastructure-as-a-Service (IaaS). This model is very flexible, but users must handle all operational management of the database system themselves. An emerging trend for such services is to exploit innovations such as "spot pricing" in the underlying IaaS services to optimize costs for non-critical workloads. In contrast to IaaS, managed services, offered either by first-party cloud providers or third-party multi-cloud vendors, sharply reduce operational complexity but provide less flexibility. When managed services were introduced, users paid for them by a provisioned capacity model. Alternative consumption models have now emerged: usage-based pricing as well as hybrid models that support on-demand event-driven auto-scaling of compute and storage. As we continue to move away from pre-provisioned resources to on-demand elastic infrastructure, including serverless, new challenges for state management arise. What will be the best way to offer serverless database services with the pay-as-you-go on-demand model? Such event-driven on-the-fly creation of data services will have a significant impact on the architecture of the query or the storage engines. Two other key difficulties for users of cloud data services are the absence of SLAs for cloud data services that go beyond availability and the lack of transparency on how auto-scaling and other choices affect the cost.

Challenges of cloud architecture: The cloud architecture raises unique opportunities and challenges for innovative database system design:

- *Disaggregation*: An important characteristic of cloud architectures is the use of very large pools of commodity hardware that are subject to hardware and software failures at scale. To handle such failures, modern cloud databases are increasingly decoupling storage and compute for high availability, scalability, and durability. For example, all distributed data lake platforms have been or are being rearchitected with compute and storage services decoupled. Disaggregation is key to enabling elastic compute, but its ability to meet response time requirements critically depends on caching effectively, which is challenging due to the multiple levels of memory hierarchy. It also depends on supporting some minimal compute within the storage service that can sharply reduce data movement. (See also Section 3.4)

- *Multi-tenancy:* Unlike traditional environments where resources are scarce and carefully provisioned per workload, cloud-based infrastructure offers an opportunity to rethink databases in a world with an abundance of resources that can be pooled together for a set of workloads. In such an environment, it is critical to support multi-tenancy to control costs and utilization. This requires mechanisms to respond swiftly and alleviate resource pressure as demand causes local spikes. Telemetry can be used to predict usage and take proactive measures. Over longer time frames, there are challenges of capacity management. The range of required innovation here spans reimagining database systems as composite single-tenant and multi-tenant microservices, creating and operating predictive models, developing mechanisms for agile response to resource demands, reorganizing resources among active tenants dynamically without impacting active application workloads and ensuring tenants are isolated from noisy neighbor tenants.

- *Hybrid cloud*: In an ideal world, on-premise data platforms would seamlessly draw upon compute and storage resources available in the cloud "on-demand". There is a pressing need to identify architectural approaches that make it possible for on-premise data infrastructure and cloud systems to take advantage of each other instead of relying on "cloud only" or "on-premise only". As enterprises split their data processing across on-premise systems and the cloud, they need a single control plane for the entire data estate.

- *Edge and cloud*: IoT has resulted in a skyrocketing number of computing devices connected to the cloud, in some cases only intermittently. The limited capabilities of these devices, characteristics of their connectivity (e.g., limited bandwidth for offshore devices, ample bandwidth for 5G-connected devices), and their data profiles will lead to new optimization challenges for distributed data processing and analytics.

Leveraging uniqueness of SaaS: Software-as-a-Service (SaaS) applications need to be multi-tenant. But in contrast to ad-hoc multi-tenancy, each tenant has approximately or exactly the same database schema (but no shared data) and the same application code. One way to support multi-tenant SaaS applications is to have all tenants share one database instance with the logic to support multi-tenancy pushed into the application stack. While this is simple to support from a database platform perspective, this approach makes customization (e.g., schema evolution), query optimization, and insulation from a noisy neighbor harder. The other extreme approach is to spawn a separate database instance for each tenant. While flexible, this approach is not cost-effective as it fails to take advantage of commonality among tenants. Yet another approach is to pack tenants into shards with large tenants placed in shards of their own. Such packing of tenants into shards is nontrivial. Moreover, security considerations may also constrain the specific architecture that is chosen. Thus, there is a need to think carefully about tradeoffs between design alternatives in architecting SaaS and about functionality that needs to be supported at the cloud database infrastructure vs. implemented in the application stack.

Multi data center issues: Cloud applications that operate across multiple data centers, potentially geographically distant from each other, remain a key challenge for both analytics and active-active online transaction processing (OLTP) workloads (see details in Section 3.4). Some countries have data sovereignty laws that make it illegal to move their citizens' data to another country's data center. More work is needed to understand how these considerations impact data center replication and high availability.

Auto-tuning: Cloud databases need to support a diverse set of time-varying multi-tenant workloads, and no one configuration tuning works well universally. Furthermore, the vastly expanded user base of cloud databases lacks expert DBAs. Studies of cloud workloads indicate that many cloud database applications do not use best practices for configuration settings, schema design, or data access code. Thus, while auto-tuning has always been important, for cloud databases it is of critical importance. Fortunately, cloud systems' telemetry logs are plentiful and present a great opportunity to improve the auto-tuning functionality of the database systems. Machine learning may be helpful here as well.

Confidential cloud computing: Enterprises are concerned about security and privacy of their data when it moves into a public cloud. This has led to the rise of confidential cloud computing, which makes data visible only to the enterprise, so no security lapse in the cloud infrastructure compromises its privacy. The challenge is to enable this functionality with an acceptable loss of performance. While there has been progress in this area, a comprehensive approach to confidential cloud computing is yet to emerge and thus this remains a fertile research area.

Opportunity for data sharing: The cloud offers a unique opportunity for flexible data sharing. More importantly, we need to define architectures for data sharing idioms. In its most stringent form, data sharing can be viewed as a variant of the multi-party computation (related to confidential cloud computing described above). In its simplest form, it is the ability to leverage public data sets along with private data sets. We should explore other idioms of data sharing between these extremes. A related challenge is a scalable flexible search for data sets and providing provenance and other related metadata that are crucial for data sharing.

Minimizing lock-in for cloud data services and facilitating interoperability across multi-clouds will benefit all users. Today, each public cloud is a

"walled garden", created using rich supporting tools around data services. While important, there has been little effort in facilitating multi-cloud either in the industry or in the research community.

3.4 Database Engines

As mentioned earlier in the report, the last two decades have witnessed significant changes that have impacted the architecture of data platforms. One change affecting the core database engines is the rise of scalable distributed "document-stores", which support key-value look up and horizontal scaling. Another is the evolution of the Hadoop ecosystem to a more efficient Spark ecosystem for extract-transform-load jobs (ETL) and support for relational execution over such a runtime by leveraging query processing techniques from database engines. New memory-optimized data structures, compilation, and code-generation have significantly enhanced performance of traditional database engines. Main-memory database techniques have become established in both industry and research, often as part of HTAP systems. Another key achievement of the field are highly scalable streaming systems that are widely used. All data analytics engines have now implemented column-oriented storage. The cloud has reinvigorated work in geo-distributed replication, and the industry has made significant progress on that front. As mentioned earlier, the need for elastic computation in cloud platforms has led to rearchitecting database engines for disaggregated storage and compute.

We now discuss the key themes related to the evolution of database engines.

Heterogeneous computation: We see a clear trend towards heterogeneous computation with the death of Dennard scaling and the advent of new accelerators introduced to offload compute. GPUs and FPGAs are available today, with the software stack for GPUs much better developed than that for FPGAs. Likewise, we see increasing deployments of RDMA. The memory and storage hierarchy is also more heterogeneous than ever before. The advent of high-speed SSDs has already had significant performance impact and altered the traditional trade-offs between in-memory systems and disk-based database engines. Engines with the new generation of SSDs are destined to erode some of the key benefits of in-memory systems. Furthermore, NVRAM is finally becoming generally available, which might have significant impact on database engines due to their support for persistence and low latency. Embracing this new normal world of heterogeneous hardware and re-architecting the database engine accordingly will be one of the most important tasks for the community. We also need to explore what would be an ideal hardware-software co-design that will best support database engines. For example, a co-design could help with asynchronously doing bulk lookup of objects for queries. Thus, we expect database architects to have an active agenda to take advantage of disaggregation, recent and upcoming hardware innovations, and to explore hardware-software co-design.

Data lakes and modern data warehousing applications: The needs of traditional data warehousing applications have expanded. They need to consume data from a variety of data sources. They need to transform the data and perform complex analyses more quickly. These new requirements have a profound impact on the design of core database engines that support them. The community is in the middle of a transition from classical data warehouses to a data-lake-oriented architecture for analytics. Although popularized in the public cloud due to the wide availability of scalable low-cost blob storage, a data lake architecture is equally applicable for on-premise systems. Instead of a traditional setting where data is ingested into an OLTP store and then swept into a data warehouse through an ETL process, perhaps powered by a Big Data framework such as Spark, the data lake is a flexible repository that can ingest a variety of data objects. Subsequently, a variety of compute engines can operate on the data, to curate it or execute complex SQL queries, and store the results back in the data lake or send it to other operational systems. Thus, data lakes exemplify a disaggregated architecture. One unique challenge of data lakes is *scalable data discovery*. Therefore, *data profiling*, which provides a statistical characterization of data, is of utmost importance in data lakes. Data profiling is challenging for data lakes, as profiling must have low latency despite having to provide a statistical summary of very large, heterogeneous, and possibly semi-structured data sets. Other challenges include finding all data relevant to a task quickly, e.g., identifying data that is joinable with other relevant data sets after suitable transformations.

Leveraging approximation: As the volume of data continues to explode, we must seek techniques that reduce latency or increase throughput of query processing. For example, leveraging approximation for *fast progressive visualization* of answers to queries

over data lakes can help exploratory data analysis to unlock insights in data. Data sketches have already gone mainstream and are classic examples of effective approximations. Sampling is another tool that can be used to reduce the cost of query processing. However, the support for sampling in today's Big Data systems is quite limited and does not cater to the richness of query languages such as SQL. Our community has done much foundational work in approximate query processing, but we need a better way to expose it in a programmer-friendly manner with clear semantics.

Distributed transactions: Data management systems are increasingly being distributed across multiple machines both within a single region and across multiple geographic regions. This has renewed interest in industry and academia on the challenges of processing distributed transactions. The increased complexity and variability of failure scenarios, combined with increased communication latency and performance variability in distributed architectures has resulted in a wide array of tradeoffs between consistency, isolation level, availability, latency, throughput under contention, elasticity, and scalability. There is an ongoing debate between two schools of thought: (1) Distributed transactions are hard to process at scale with high throughput and availability and low latency without giving up some traditional transactional guarantees. Therefore, consistency and isolation guarantees are reduced at the expense of increased developer complexity. (2) The complexity of implementing a bug-free application is extremely high unless the system guarantees strong consistency and isolation. Therefore, the system should offer the best throughput, availability, and low-latency service it can, without sacrificing correctness guarantees. This debate will likely not be fully resolved anytime soon, and industry will offer systems consistent with each school of thought. However, it is critical that application bugs and limitations in practice that result from weaker system guarantees be better identified and quantified, and tools be built to help application developers using both types of system achieve their correctness and performance goals.

Leveraging machine learning: Recent advances in ML have inspired our community to reflect on how some of hard data engine problems could use ML to advance the state of the art. The most obvious such problems are in *auto tuning*. For example, we can systematically replace "magic numbers" in database systems with data-driven learning models and use them to auto-tune system configurations. ML also provides new hope for progress in query optimization, which has seen only minor improvements in the last two decades. Although in principle almost any component can potentially be improved with ML, answers to some key questions are prerequisites for success, such as availability of training data, a well-thought software engineering pipeline to support an ML component (debuggability is notoriously hard), and availability of the guard-rails so that when test data or test queries deviate from the training data and training queries, the system degrades gracefully.

Support for machine learning in database engines: As we briefly discussed in Section 3.1, modern data management workloads include ML. This adds an important, new requirement for database engines. We must immediately address the challenge of efficiently supporting "in-database" ML. Today, this is achieved by leveraging the traditional extensibility mechanisms of databases. However, as DNN models become more popular and bigger, supporting efficient inferencing and training will require database engines to leverage heterogeneous hardware and support popular ML programming frameworks. This evolution is still in its early stage. Database engine architects need to work with architects responsible for building ML infrastructure using FPGAs, GPUs and specialized ASICs.

Benchmarking: Over the years, benchmarks tremendously helped move forward the database industry and the database research community. The traditional benchmarks (e.g., TPC- E, TPC-DS, TPC-H) that the database community has developed are good, but they do not capture the full breadth and depth of our field. We need to reaffirm our commitment to invest in benchmarking for the new application scenarios and database engine architectures. For example, without development of appropriate benchmarking and data sets, a fair comparison between traditional database architectures and ML-inspired architectural modifications to database systems will be impossible. The community needs new benchmarks that capture the needs of modern workloads, in particular with respect to the velocity and variety dimensions of Big Data, e.g., streaming scenarios, data with skew, workloads with common data transformations, and processing of new types of data such as videos. While some benchmarks are emerging in this space, much more remains to be done. A closely related issue is that of poor practice of performance evaluation in publications. The

selection of workloads, databases, and parameters often lacks rigor. Moreover, only a simplistic aggregate metric (average) is typically reported, instead of providing important additional metrics such as the variance.

SQL Standard: While the SQL Standard has been a major benefit to the ecosystem, SQL implementations in different data systems still differ in semantics. Our community must continue to push towards making SQL a true standard. At the same time, SQL may not be enough for supporting data science applications and ML workloads. Therefore, we need to investigate systems that combine relational algebra and linear algebra in a richer query paradigm, potentially as extensions to SQL.

Two "holy grails" should continue to stay on our agenda. First, we must always explore any novel ideas to reduce the impedance mismatch between application development and writing database queries. Second, we must continue to find ways to make database systems less rigid (e.g., flexible schema evolution) without significantly sacrificing their performance.

4. COMMUNITY

The database community is in a healthy state with stable numbers of submissions to our conferences. Our flagship conferences are well attended by academics and engineers from the industry. Nevertheless, the community continues to have great opportunities to improve, as discussed below.

End-to-end solutions in the hands of users: To increase its impact, the database research community should put more weight on developing (or participating in the building of) full-fledged systems, as well as use these systems and tools to help real users. Incorporating the algorithmic innovations from our community into working systems will greatly increase their impact and reach. Moreover, by interacting with real users, the community will increase its impact, visibility, and connection with today's rapidly changing data management challenges.

Open source and Cloud Services: To achieve high impact, our community should develop tools that are easy to adopt. This will be the case if they are part of existing, popular ecosystems of open-source tools or are available as easy-to-use cloud services. Specifically, an important way through which we can have impact is when we rally around large-scale open-source systems. Such systems accelerate innovations because new algorithmic ideas can be readily added to them. They also become mature tools, with large communities of developers that can more easily support real users. Recent examples of such systems that either came out (or, included significant input from) the database community include Apache Spark, Apache Flink, and Apache Kafka.

Data science software ecosystem: The database community must do a better job integrating database research with the data science ecosystem (e.g., Jupyter notebooks, Python). Database techniques for data integration, data cleaning, data processing, and data visualization should be easily called from Python scripts. The community can also develop more elegant APIs that scale more naturally and better integrate relational and linear algebras. These tools need to work well at any scale, not just for the biggest problems. Users should be able to try the tools at small scale on their laptops or in the cloud and should not have to make significant changes to their code as their data and computational needs grow.

Community innovation: The database community has always been innovating its approach to running conferences, such as having multiple conference deadlines per year, changing how papers are assigned and reviewed, pioneering the evaluation of the reproducibility of accepted papers, expanding the conference programs to include tutorials. Going forward, it is important for the community to remain innovative and to continue to improve the paper selection processes. Several factors have surfaced in recent years. First, database researchers now have tremendous opportunities for ambitious research projects, large centers, and exciting collaborations with industry. As a result, many researchers are too busy to participate in program committees as often as they would like, even for our top conferences, because they are simply stretched too thin. We need to find a solution to this problem because we want the input of all our community members in the reviewing process. Second, paper reproducibility continues to be a challenge. The community has experimented with many techniques to encourage researchers to make their work open-source and reproducible, but this continues to be an uphill battle. The community needs to continue to explore innovative ideas to remedy this problem. Third, the current review process and common understanding among reviewers honors algorithmic research contributions as "syntactically correct papers", with a clear definition of a baseline that an

algorithm improves upon, whereas some members of the community do not sufficiently value other contributions, such as papers describing innovative systems, applications, experiments and analyses, or user studies. One way to improve the situation is to more strongly emphasize "potential for impact" in the reviewing process.

Impact on university campuses: The advent of data science and the excitement around data science education, including data science minors, majors, master's degrees, and specializations within existing majors, are great opportunities for the database community to broadly influence education on our university campuses. Data science is multidisciplinary. It includes components from machine learning, human computer interaction, statistics, ethics, law, data visualization, application domains, and of course data management. Many aspects of data management are very relevant to data science, as discussed in Section 3.1 of this report. Students from all fields of study (not only computer science) need to learn the technology our community has developed over the years to support the data to insights pipeline. Therefore, database experts are natural collaborators for teaching data science and devising data science curricula. Database faculty should thus be engaged in discussions that define the data science curriculum on their respective campuses.

5. LOOKING FORWARD

It is impossible to capture fully the exciting discussions we have had in our meeting in Fall 2018. This report summarizes some of the key recommendations and reflections from that meeting. A downloadable copy of this report as well as some supplementary materials used in the meeting can be found on the event website [1].

Beyond the summary recommendations in this report, database researchers should be attentive to technology and application trends. Much has already changed since our Fall 2018 meeting. Every new mechanism that has emerged offers a potential opportunity to enhance data management capabilities (e.g., blockchain, quantum computing) and every new scenario is a potential application area where data management might help (e.g., self-driving cars, fake news).

As the database community continues its strong history of impact on research and industry, we are fortunate to have many exciting research directions around data science, machine learning, data governance, as well as new architectures for cloud systems and data engines. We need to focus on building more impactful open-source software systems and cloud services based on our research, and better integrate with the existing data science stack and tools. In our conferences, we must revisit how scholarship is evaluated and recognized so that we are set up for maximal impact.

6. ACKNOWLEDGMENTS

The Seattle database meeting was supported financially by donations from Google, Megagon Labs, and Microsoft Corporation.

7. REFERENCES

[1] The Database Research Self-Assessment Meeting 2018. http://seattle-dbresearch-assessment.cs.washington.edu, 2018.

[2] D. Abadi, R. Agrawal, A. Ailamaki, M. Balazinska, P. A. Bernstein, M. J. Carey, S. Chaudhuri, J. Dean, A. Doan, M. J. Franklin, J. Gehrke, L. M. Haas, A. Y. Halevy, J. M. Hellerstein, Y. E. Ioannidis, H. V. Jagadish, D. Kossmann, S. Madden, S. Mehrotra, T. Milo, J. F. Naughton, R. Ramakrishnan, V. Markl, C. Olston, B. C. Ooi, C. Ré, D. Suciu, M. Stonebraker, T. Walter, and J. Widom. The Beckman Report on Database Research. *CACM*, 59(2):92–99, 2016.

[3] P. Bernstein, U. Dayal, D. DeWitt, D. Gawlick, J. Gray, M. Jarke, B. Lindsay, P. Lockemann, D. Maier, E. Neuhold, A. Reuter, L. Rowe, H.-J. Schek, J. Schmidt, M. Schrefl, and M. Stonebraker. Future Directions in DBMS Research – The Laguna Beach Participants. *ACM SIGMOD Record*, 18:17–26, 03 1989.

Digital Libraries: Supporting Open Science

Report on the 15th Italian Research Conference on Digital Libraries, IRCDL2019

Paolo Manghi, Leonardo Candela, Emma Lazzeri
ISTI - Consiglio Nazionale delle Ricerche
Pisa, Italy
{manghi, candela, lazzeri}@isti.cnr.it

Gianmaria Silvello
University of Padova
Padua, Italy
silvello@dei.unipd.it

1. INTRODUCTION

The Italian Research Conference on Digital Libraries (IRCDL) is the annual Italian forum to discuss research topics on Digital Libraries and related technical, practical, and social issues. Along the years, IRCDL touched several aspects underlying the "Digital Library" domain and promptly adapted to the evolution of the field. Today, the "Digital Library" field includes theory and practices reflecting the evolution of the role of libraries in the scholarly communication domain, and also embracing scholarly communication and open science.

The theme of IRCDL 2019 was "Digital Libraries: Supporting Open Science". Three main reasons motivated this theme: (*i*) science is increasingly becoming digital, meaning that research is performed using data services and digital tools; (*ii*) the results of the research are no longer just traditional scientific publications; (*iii*) the outcomes of science are increasingly encompassing datasets, software, and experiments. As digital artifacts, such products can be shared and re-used together with the article, thus enabling comprehensive research assessment and various degrees of reproducibility of science. Positive consequences of this shift towards Open Science are: accelerating science, optimizing the cost of research, fraud detection, and fully-fledged scientific reward. Digital Libraries are central in the evolution of research outputs by targeting findability, preservation, interlinking, and re-use of research products and by integrating the components of the scholarly communication process.

The conference has been organized in Pisa, and the proceedings are published in the Springer CCIS series Vol. 988 [22]. Pre-print versions, research datasets, and research software relative to the accepted contributions are accessible via Zenodo.org.[1]

[1] https://zenodo.org/communities/ircdl

2. CONFERENCE CONTRIBUTIONS

All submitted contributions were peer-reviewed by three of the thirty-two members of the Program Committee, and twenty-one were accepted, out of which six were short papers. IRCDL comprised of one invited speaker and six sessions.

Invited talk: Citation in the Era of Big Data and Open Source Software.

Prof. Susan B. Davidson, Weiss Professor at the Dept. of Computer and Information Science of the University of Pennsylvania, USA, discussed the most recent developments in data and software citation. Citations are the cornerstone of knowledge propagation in science and the principal means to assess the quality of research as well as to direct investments in science. We are transitioning towards the fourth paradigm of science where data and software are as vital to scientific progress as traditional publications are. Nevertheless, there is no viable computational method for citing data and software. Thus, to recognize the scientific contribution of developers, data scientists, data curators, and data centers and to estimate the value of data. Prof. Davidson presented the main challenges, and the solutions the database and digital library communities are supplying [11].

Open Science and Open Access. This session discussed on the issues originating from enacting Open Access and Open Science principles to the general public and the research world. Lana [19] advocated how Information Literacy needs Open Access, for the citizens to freely access high-quality information. Beamer [5] presented a methodology to optimize the embracing of Open Science practices in academic libraries. Fontanin [14] highlighted the Open Access-related barriers – e.g., technical infrastructures, points of access, digital and cultural di-

vide – making the information potentially available not just to researchers, but to everyone.

Open Science publishing and scientific workflows. The contributions in this session dealt with methodologies, practices, and tools in support of publishing workflows respecting Open Science principles. Latif [20] presented the work on EconStor, ZBW's Open Access Repository, to enrich attribution metadata by linking to external authority data sources. Dosso [12] described the "Learning to Cite" framework, for the creation of citation models to automatically cite XML files and its application with a process of transfer learning in the archival domain. Mizzaro [28] introduced an open-source software solution for the implementation of crowdsourcing Peer Review methodologies. Minelli [24] showcased the practical application of the open scientific life-cycle model proposed by the EcoNAOS (Ecological North Adriatic Open Science Observatory System) project. Bardi [4] illustrated a framework for the description, and peer review of research flows developed in the OpenUp project.

Text mining. Text mining techniques play a crucial role in Digital Libraries to automatically extract information used to serve user's needs better. Serra [26] proposed an approach to keyphrase extraction via an Attentive Model, a neural network designed to focus on the most relevant parts of data. Carducci [7] presented a system combining standard and semantic learning for automatically annotating bibliographic records. Pandolfo [25] described how they built the semantic layer of the Piłsudski Institute of America digital archive. Ferilli [13] described the work performed to extend the BLA-BLA tool for learning linguistic resources by adding a Grammar Induction feature based on the advanced process mining and management system WoMan. Petrocchi [9] presented a study performed on Google Shopping to showcase how large search engines apply query steering depending on the user's profile.

Research Communities and Research Data. Research communities and the way they manage research data are increasingly becoming critical elements of digital libraries. Witt [31] presented the Repository Finder tool, designed to help researchers in the domain of Earth, space, and environmental sciences at finding the thematic repository they need based on a user-friendly wizard. Vezzani [30] presented TriMED, a digital library of terminological records designed to satisfy the information needs of different categories of users within the healthcare field. Castro described the results of two exploratory studies: in [27] the authors adopt a researcher-curator collaborative approach involving researchers in metadata description and discussing the use of generic and domain-oriented metadata; in [17] the authors analyze a data deposition workflow in CKAN using a Dublin Core metadata model for non-expert users. Luzi and Ruggieri [21] presented the OpenUp project pilot on research data sharing, validation, and dissemination in Social Sciences, intending to investigate the applicability of peer review and/or Open Peer Review to datasets in disciplines related to Social sciences.

Information retrieval and discovery. The relationship between information retrieval and discovery with digital libraries is long-standing. Fabris [1] presented a study exploring the relationships between SIGIR Information Retrieval articles from 2003 to 2017 with topics in the Digital Library domain. The goal is to identify trends and synergies between the two research fields. Amelio [2] showcased a study of the CAPTCHA usability which analyses the predictability of the solution time, also called response time, to solve the Dice CAPTCHA and suggested strategies towards the achievement of the "optimal" CAPTCHA. Tardelli [10] introduced on-demand tools provided by the `SoBigData.eu` research infrastructure for user-driven monitoring of Twitter data and publishing of the results as research data. Hast [16] described a training-free word spotting algorithm to mine images of digitized historical handwritten material to enable text search across the collection. Metilli [23] presented a case-study based on the Wikidata knowledge base exploring techniques to improve search functionalities by semi-automatically extracting narratives.

Applications. The last session included contributions about four application use-cases. Mannocci [18] presented DOIBoost, a version of the CrossRef metadata collection enriched with ORCID and the Microsoft Academic Graph, and Unpaywall made public in Zenodo.org, together with the software required to generate it. Foufoulas [15] presented user interfaces included in the Research Community Dashboard service of OpenAIRE enabling users to fine-tune text mining algorithms over a 10M full-texts corpus. Bellotto and Bettella [6] illustrated the experience of extending the metadata model of the Phaidra repository (University of Wien) towards the MODS data model. Firmani and Nieddu [3] reported on the Codice Ratio project, deliver-

ing a system taking advantage of character segmentation to support paleographers with tools for the minimal-effort transcription of large medieval manuscripts from the Vatican Secret Archives.

3. CONCLUSION AND PROSPECT

The research activities and results presented at IRCDL2019 give a clear indication of how active and multifaceted Digital Library research is.

A panel of experts[2] was organized to start a dialogue aiming at identifying research directions. Digital Libraries have always supported two phases of science, namely sharing of "mature" research products and discovery of published research products. Open Science has *de facto* revolutionized this model that conceptually separated the production of science from the publishing of science. For example, Research Infrastructures offer services constituting the "digital laboratory" where scientists are executing their experiments while accessing and sharing their intermediate results with others.

Two decades ago, the DELOS Grand Vision of Digital Libraries challenges focused on "[. . . enabling] any citizen to access all human knowledge anytime and anywhere, in a friendly, multimodal, efficient and effective way, by overcoming barriers of distance, language, and culture and by using multiple Internet-connected devices" [29]. The advent of Open Science, together with the natural evolution towards digital science, has profoundly impacted on this vision. IRCDL2019 conference has widely proven this statement, by highlighting strong interests in connecting digital library methods, tools, and services with thematic services for science and Open Science challenges. The current scenario, although addressing the urgent requirements of digital science (e.g. big research data, data-intensive science, multi-disciplinarity), suffers from the downsides arising when solutions originate from spontaneous initiatives rather than overarching engineering. The scholarly record is today kept in highly distributed and poorly connected sources, operated by publishers, research infrastructures, and institutions, adhering to heterogeneous publishing workflows, publishing best practices, and standards.

As remarked by Dr. C. Thanos in the final conference panel on the Future of Digital Library research, digital library research should envision "a world in which all scientific literature, data and other research outcomes are on-line, open and interoperable [. . . and seek for . . .] the creation of discipline-specific and interdisciplinary interconnected scholarly information spaces [. . . altogether forming a global . . .] Scholarly Record". Literature, datasets, software, and other digital assets of science should reside in resource-specific digital libraries (archives, repositories, databases), intended as active nodes in scholarly infrastructures [8]. To this aim, Digital Libraries should act as critical elements of Research Infrastructures and Open Cyber-Scholarly Communication Infrastructures, therefore flexibly adapt to support scientific communities at performing and publishing science by managing any research asset. In summary, Digital Libraries have upgraded their vest, their original intent, and are evolving to serve different actors. They should ambitiously act as an enabling service between scientists performing science, scientists publishing science, scholars, and scientists discovering scientific results, innovators accessing science for industrial benefits, and officers in need of monitoring science.

[2] https://ircdl2019.isti.cnr.it/?page_id=371

Acknowledgments

We desire to thank all those who contributed to the event, especially our colleagues Costantino Thanos and Maristella Agosti for their advice. Special thanks are also due to the members of the program committee whose research experience largely contributed to making this conference an attractive venue and valuable experience. Our sincere gratitude goes to all participants, invited speakers and authors, whose enthusiasm and vision constitute the soul of this conference. This event was supported by OpenAIRE-Advance project (EU H2020 research and innovation program, grant agreement No. 777541).

4. REFERENCES

[1] M. Agosti, E. Fabris, and G. Silvello. On synergies between information retrieval and digital libraries. In Manghi et al. [22], pp 3–17.

[2] A. Amelio, R. Janković, D. Tanikić, and I. R. Draganov. Predicting the usability of the dice captcha via artificial neural network. In Manghi et al. [22], pp 44–58.

[3] S. Ammirati, D. Firmani, M. Maiorino, P. Merialdo, and E. Nieddu. In codice ratio: Machine transcription of medieval manuscripts. In Manghi et al. [22], pp 185–192.

[4] A. Bardi, V. Casarosa, and P. Manghi. Foundations of a framework for peer-reviewing the research flow. In Manghi et al. [22], pp 195–208.

[5] J. E. Beamer. Digital libraries for open science: Using a socio-technical interaction network approach. In Manghi et al. [22], pp 122–129.

[6] A. Bellotto and C. Bettella. Metadata as semantic palimpsests: The case of PHAIDRA@UNIPD. In Manghi et al. [22], pp 167–184.
[7] G. Carducci, M. Leontino, D. P. Radicioni, G. Bonino, E. Pasini, and P. Tripodi. Semantically aware text categorisation for metadata annotation. In Manghi et al. [22], pp 315–330.
[8] D. Castelli, P. Manghi, and C. Thanos. A vision towards scientific communication infrastructures - on bridging the realms of research digital libraries and scientific data centers. *Int. J. on Digital Libraries*, 13(3-4):155–169, 2013.
[9] V. Cozza, V. T. Hoang, M. Petrocchi, and R. De Nicola. Transparency in keyword faceted search: An investigation on google shopping. In Manghi et al. [22], pp 29–43.
[10] S. Cresci, S. Minutoli, L. Nizzoli, S. Tardelli, and M. Tesconi. Enriching digital libraries with crowdsensed data. In Manghi et al. [22], pp 144–158.
[11] S. B. Davidson, P. Buneman, D. Deutch, T. Milo, and G. Silvello. Data Citation: A Computational Challenge. In *Proc. of the 36th ACM PODS 2017*, pp 1–4, 2017.
[12] D. Dosso, G. Setti, and G. Silvello. Learning to cite: Transfer learning for digital archives. In Manghi et al. [22], pp 97–106.
[13] S. Ferilli and S. Angelastro. Towards a process mining approach to grammar induction for digital libraries. In Manghi et al. [22], pp 291–303.
[14] M. Fontanin and P. Castellucci. Water to the thirsty reflections on the ethical mission of libraries and open access. In Manghi et al. [22], pp 61–71.
[15] T. Giannakopoulos, Y. Foufoulas, H. Dimitropoulos, and N. Manola. Interactive text analysis and information extraction. In Manghi et al. [22], pp 340–350.
[16] A. Hast, P. Cullhed, E. Vats, and M. Abrate. Making large collections of handwritten material easily accessible and searchable. In Manghi et al. [22], pp 18–28.
[17] Y. Karimova, J. A. Castro, and C. Ribeiro. Data deposit in a ckan repository: A dublin core-based simplified workflow. In Manghi et al. [22], pp 222–235.
[18] S. La Bruzzo, P. Manghi, and A. Mannocci. Openaire's doiboost - boosting crossref for research. In Manghi et al. [22], pp 133–143.
[19] M. Lana. Information literacy needs open access or: Open access is not only for researchers. In Manghi et al. [22], pp 236–247.
[20] A. Latif, T. Borst, and K. Tochtermann. Collecting and controlling distributed research information by linking to external authority data - a case study. In Manghi et al. [22], pp 331–339.
[21] D. Luzi, R. Ruggieri, and L. Pisacane. The openup pilot on research data sharing, validation and dissemination in social sciences. In Manghi et al. [22], pp 248–258.
[22] P. Manghi, L. Candela, and G. Silvello, editors. Proc. of the *15th Italian Research Conference on Digital Libraries, IRCDL 2019*, CCIS 988, Springer, 2019.
[23] D. Metilli, V. Bartalesi, C. Meghini, and N. Aloia. Populating narratives using wikidata events: An initial experiment. In Manghi et al. [22], pp 159–166.
[24] A. Minelli, A. Sarretta, A. Oggioni, C. Bergami, and A. Pugnetti. A practical workflow for an open scientific lifecycle project: Econaos. In Manghi et al. [22], pp 209–221.
[25] L. Pandolfo, L. Pulina, and M. Zieliński. Exploring semantic archival collections: The case of piłsudski institute of america. In Manghi et al. [22], pp 107–121.
[26] M. Passon, M. Comuzzo, G. Serra, and C. Tasso. Keyphrase extraction via an attentive model. In Manghi et al. [22], pp 304–314.
[27] J. Rodrigues, J. A. Castro, J. R. da Silva, and C. Ribeiro. Hands-on data publishing with researchers: Five experiments with metadata in multiple domains. In Manghi et al. [22], pp 274–288.
[28] M. Soprano and S. Mizzaro. Crowdsourcing peer review: As we may do. In Manghi et al. [22], pp 259–273.
[29] C. Thanos and V. Casarosa. The key role of the DELOS Network of Excellence in establishing Digital Libraries as a research field in Europe. *LIBEER Quarterly*, 26(4):296–307, 2017.
[30] F. Vezzani and G. M. Di Nunzio. Computational terminology in ehealth. In Manghi et al. [22], pp 72–85.
[31] M. Witt, S. Stall, R. Duerr, R. Plante, M. Fenner, R. Dasler, P. Cruse, S. Hou, R. Ulrich, and D. Kinkade. Connecting researchers to data repositories in the earth, space, and environmental sciences. In Manghi et al. [22], pp 86–96.