

# COMMUNICATIONS

CACM.ACM.ORG

OF THE  
**ACM**

07/2016 VOL.59 NO.07



## THE RISE OF SOCIAL BOTS

EDITORIAL THE RITUAL OF ACADEMIC-UNIT REVIEW NEWS ACCELERATING SEARCH  
VIEWPOINTS INVERSE PRIVACY CONTRIBUTED ARTICLES FORMULA-BASED DEBUGGING  
RESEARCH HIGHLIGHTS GOOGLE'S MESA DATA WAREHOUSING SYSTEM

**JULY 1**

Association for  
Computing Machinery



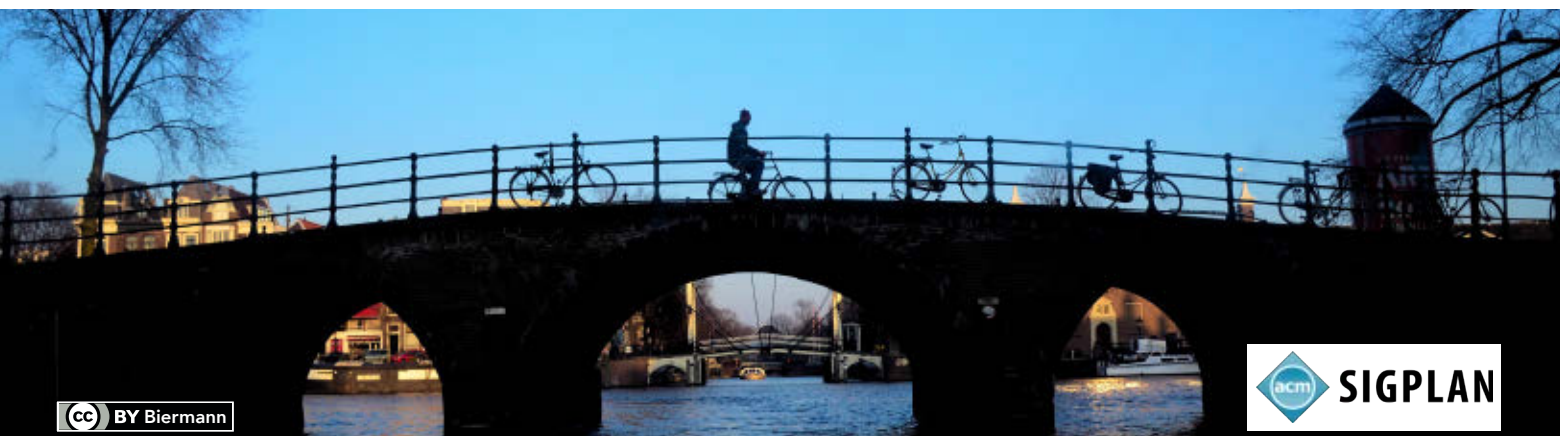
# this.splash 2016

Amsterdam

Sun 30 October – Fri 4 November 2016

## ACM SIGPLAN Conference on Systems, Programming, Languages and Applications: Software for Humanity (SPLASH)

|                 |                                                                                     |
|-----------------|-------------------------------------------------------------------------------------|
| <b>OOPSLA</b>   | Novel research on software development and programming                              |
| <b>Onward!</b>  | Radical new ideas and visions related to programming and software                   |
| <b>SPLASH-I</b> | World class speakers on current topics in software, systems, and languages research |
| <b>SPLASH-E</b> | Researchers and educators share educational results, ideas, and challenges          |
| <b>DLS</b>      | Dynamic languages, implementations, and applications                                |
| <b>GPCE</b>     | Generative programming: concepts and experiences                                    |
| <b>SLE</b>      | Principles of software language engineering, language design, and evolution         |



**SPLASH General Chair:** Eelco Visser

**OOPSLA Papers:** Yannis Smaragdakis

**OOPSLA Artifacts:** Michael Bond, Michael Hind

**Onward! Papers:** Emerson Murphy-Hill

**Onward! Essays:** Crista Lopes

**SPLASH-I:** Eelco Visser, Tijs van der Storm

**SPLASH-E:** Matthias Hauswirth, Steve Blackburn

**DLS:** Roberto Ierusalimsky

**Workshops:** Jan Rellermeier, Craig Anslow

**SLE General Chair:** Tijs van der Storm

**SLE Papers:** Emilie Balland, Daniel Varro

**GPCE General Chair:** Bernd Fischer

**GPCE Papers:** Ina Schaefer

**Student Research Competition:** Sam Guyer, Patrick Lam

**Posters:** Jeff Huang, Sebastian Erdweg

**Publications:** Alex Potanin

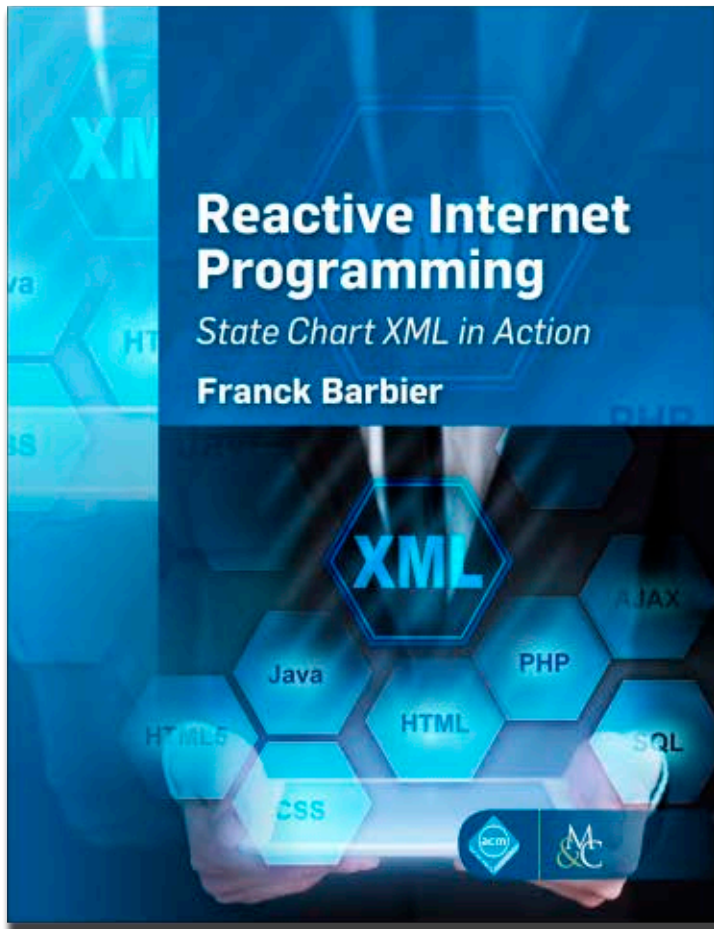
**Publicity and Web:** Tijs van der Storm, Ron Garcia

**Student Volunteers:** Daco Harkes

 @splashcon

2016.splashcon.org

 bit.ly/splashcon16



You need effective means to put reactive programming into practice.

THEY'RE  
RIGHT HERE

Is Internet software so different from “ordinary” software? This book practically answers this question through the presentation of a software design method based on the State Chart XML W3C standard along with Java. Web enterprise, Internet-of-Things, and Android applications, in particular, are seamlessly specified and implemented from “executable models.”

Internet software puts forward the idea of event-driven or reactive programming, as pointed out in Bonér et al.’s “Reactive Manifesto”. It tells us that reactivity is a must. However, beyond concepts, software engineers require effective means with which to put reactive programming into practice. *Reactive Internet Programming* outlines and explains such means.

The lack of professional examples in the literature that illustrate how reactive software should be shaped can be quite frustrating. Therefore, this book helps to fill in that gap by providing in-depth professional case studies that contain comprehensive details and meaningful alternatives. Furthermore, these case studies can be downloaded for further investigation.

Internet software requires higher adaptation, at run time in particular. After reading *Reactive Internet Programming*, you will be ready to enter the forthcoming Internet era.



<http://books.acm.org>  
<http://www.morganclaypoolpublishers.com/xml>

## Departments

- 5 **Editor's Letter**  
**The Ritual of Academic-Unit Review**  
*By Moshe Y. Vardi*
- 
- 7 **Cerf's Up**  
**The Power of Big Ideas**  
*By Vinton G. Cerf*
- 
- 8 **Letters to the Editor**  
**Rethinking Computational Thinking**
- 
- 10 **BLOG@CACM**  
**Progress in Computational Thinking, and Expanding the HPC Community**  
Jeannette Wing considers the proliferation of computational thinking, while Dan Stanzione hopes to bring more HPC practitioners to SC16.
- 
- 29 **Calendar**
- 
- 126 **Careers**

## Last Byte

- 128 **Upstart Puzzles**  
**Chair Games**  
*By Dennis Shasha*

## News



- 12 **Graph Matching in Theory and Practice**  
A theoretical breakthrough in graph isomorphism excites complexity experts, but will it lead to any practical improvements?  
*By Neil Savage*
- 
- 15 **Accelerating Search**  
The latest in machine learning helps high-energy physicists handle the enormous amounts of data produced by the Large Hadron Collider.  
*By Marina Krakovsky*
- 
- 17 **Booming Enrollments**  
The Computing Research Association works to quantify the extent, and causes, of a jump in undergraduate computer science enrollments.  
*By Lawrence M. Fisher*
- 
- 19 **Legal Advice on the Smartphone**  
New apps help individuals contest traffic, parking tickets.  
*By Keith Kirkpatrick*

## Viewpoints

- 22 **Legally Speaking**  
**Apple v. Samsung and the Upcoming Design Patent Wars?**  
Assessing an important recent design patent infringement court decision.  
*By Pamela Samuelson*
- 
- 25 **Historical Reflections**  
**How Charles Bachman Invented the DBMS, a Foundation of Our Digital World**  
His 1963 Integrated Data Store set the template for all subsequent database management systems.  
*By Thomas Haigh*
- 
- 31 **Computing Ethics**  
**Big Data Analytics and Revision of the Common Rule**  
Reconsidering traditional research ethics given the emergence of big data analytics.  
*By Jacob Metcalf*
- 
- 34 **Viewpoint**  
**Turing's Red Flag**  
A proposal for a law to prevent artificial intelligence systems from being mistaken for humans.  
*By Toby Walsh*
- 
- 38 **Viewpoint**  
**Inverse Privacy**  
Seeking a market-based solution to the problem of a person's unjustified inaccessibility to their private information.  
*By Yuri Gurevich, Efim Hudis, and Jeannette M. Wing*



Watch the authors discuss their work in this exclusive *Communications* video.  
<http://cacm.acm.org/videos/inverse-privacy>

## Practice



44

#### 44 Should You Upload or Ship Big Data to the Cloud?

The accepted wisdom does not always hold true.

*By Sachin Date*

#### 52 The Small Batches Principle

Reducing waste, encouraging experimentation, and making everyone happy.

*By Thomas A. Limoncelli*

#### 58 Statistics for Engineers

Applying statistical techniques to operations data.

*By Heinrich Hartmann*



Articles' development led by [acmqueue.queue.acm.org](http://acmqueue.queue.acm.org)



#### About the Cover:

Social bots are populating faster than technology can be designed to detect them. While some bots are harmless, a growing faction can endanger online ecosystems. This month's cover story reviews current efforts to detect social bots on Twitter. Cover illustration by Arn0.

## Contributed Articles



88

#### 68 Formula-Based Software Debugging

Satisfiability modulo theory solvers can help automate the search for the root cause of observable software errors.

*By Abhik Roychoudhury and Satish Chandra*

#### 78 Why Google Stores Billions of Lines of Code in a Single Repository

Google's monolithic repository provides a common source of truth for tens of thousands of developers around the world.

*By Rachel Potvin and Josh Levenberg*

#### 88 $\lambda > 4$ : An Improved Lower Bound on the Growth Constant of Polyominoes

The universal constant  $\lambda$ , the growth constant of polyominoes (think Tetris pieces), is rigorously proved to be greater than 4.

*By Gill Barequet, Günter Rote, and Mira Shalah*

## Review Articles

#### 96 The Rise of Social Bots

Today's social bots are sophisticated and sometimes menacing. Indeed, their presence can endanger online ecosystems as well as our society.

*By Emilio Ferrara, Onur Varol, Clayton Davis, Filippo Menczer, and Alessandro Flammini*



Watch the authors discuss their work in this exclusive *Communications* video.  
<http://cacm.acm.org/videos/the-rise-of-social-bots>

## Research Highlights

#### 106 Technical Perspective

##### Combining Logic and Probability

*By Henry Kautz and Parag Singla*

#### 107 Probabilistic Theorem Proving

*By Vibhav Gogate and Pedro Domingos*

#### 116 Technical Perspective

##### Mesa Takes Data Warehousing to New Heights

*By Sam Madden*

#### 117 Mesa: A Geo-Replicated

##### Online Data Warehouse for Google's Advertising System

*By Ashish Gupta, Fan Yang, Jason Govig, Adam Kirsch, Kelvin Chan, Kevin Lai, Shuo Wu, Sandeep Dhoot, Abhilash Rajesh Kumar, Ankur Agiwal, Sanjay Bhansali, Mingsheng Hong, Jamie Cameron, Masood Siddiqi, David Jones, Jeff Shute, Andrey Gubarev, Shivakumar Venkataraman, and Divyakant Agrawal*



Association for Computing Machinery  
Advancing Computing as a Science & Profession



ACM, the world's largest educational and scientific computing society, delivers resources that advance computing as a science and profession. ACM provides the computing field's premier Digital Library and serves its members and the computing profession with leading-edge publications, conferences, and career resources.

**Executive Director and CEO**  
Bobby Schnabel  
**Deputy Executive Director and COO**  
Patricia Ryan  
**Director, Office of Information Systems**  
Wayne Graves  
**Director, Office of Financial Services**  
Darren Ramdin  
**Director, Office of SIG Services**  
Donna Cappel  
**Director, Office of Publications**  
Bernard Rous  
**Director, Office of Group Publishing**  
Scott E. Delman

#### ACM COUNCIL

**President**  
Alexander L. Wolf  
**Vice-President**  
Vicki L. Hanson  
**Secretary/Treasurer**  
Erik Altman  
**Past President**  
Vinton G. Cerf  
**Chair, SGB Board**  
Patrick Madden  
**Co-Chairs, Publications Board**  
Jack Davidson and Joseph Konstan  
**Members-at-Large**  
Eric Allman; Ricardo Baeza-Yates;  
Cherri Pancake; Radia Perlman;  
Mary Lou Soffa; Eugene Spafford;  
Per Stenström  
**SGB Council Representatives**  
Paul Beame; Jenna Neefe Matthews;  
Barbara Boucher Owens

#### BOARD CHAIRS

**Education Board**  
Mehran Sahami and Jane Chu Prey  
**Practitioners Board**  
George Neville-Niel

#### REGIONAL COUNCIL CHAIRS

**ACM Europe Council**  
Dame Professor Wendy Hall  
**ACM India Council**  
Srinivas Padmanabhuni  
**ACM China Council**  
Jiaquan Sun

#### PUBLICATIONS BOARD

**Co-Chairs**  
Jack Davidson; Joseph Konstan  
**Board Members**  
Ronald F. Boisvert; Anne Condon;  
Nikil Dutt; Roch Guerrin; Carol Hutchins;  
Yannis Ioannidis; Catherine McGeoch;  
M. Tamer Ozsu; Mary Lou Soffa; Alex Wade;  
Keith Webster

**ACM U.S. Public Policy Office**  
Renee Dopplick, Director  
1828 L Street, N.W., Suite 800  
Washington, DC 20036 USA  
T (202) 659-9711; F (202) 667-1066

**Computer Science Teachers Association**  
Mark R. Nelson, Executive Director

# COMMUNICATIONS OF THE ACM

Trusted insights for computing's leading professionals.

*Communications of the ACM* is the leading monthly print and online magazine for the computing and information technology fields. *Communications* is recognized as the most trusted and knowledgeable source of industry information for today's computing professional. *Communications* brings its readership in-depth coverage of emerging areas of computer science, new trends in information technology, and practical applications. Industry leaders use *Communications* as a platform to present and debate various technology implications, public policies, engineering challenges, and market trends. The prestige and unmatched reputation that *Communications of the ACM* enjoys today is built upon a 50-year commitment to high-quality editorial content and a steadfast dedication to advancing the arts, sciences, and applications of information technology.

#### STAFF

**DIRECTOR OF GROUP PUBLISHING**  
Scott E. Delman  
cacm-publisher@cacm.acm.org

**Executive Editor**  
Diane Crawford  
**Managing Editor**  
Thomas E. Lambert  
**Senior Editor**  
Andrew Rosenbloom  
**Senior Editor/News**  
Larry Fisher  
**Web Editor**  
David Roman  
**Rights and Permissions**  
Deborah Cotton

**Art Director**  
Andrij Borys  
**Associate Art Director**  
Margaret Gray  
**Assistant Art Director**  
Mia Angelica Balaquiot  
**Designer**  
Iwona Usakiewicz  
**Production Manager**  
Lynn D'Addesio  
**Director of Media Sales**  
Jennifer Ruzicka  
**Publications Assistant**  
Juliet Chance

**Columnists**  
David Anderson; Phillip G. Armour;  
Michael Cusumano; Peter J. Denning;  
Mark Guzdial; Thomas Haigh;  
Leah Hoffmann; Mari Sako;  
Pamela Samuelson; Marshall Van Alstyne

#### CONTACT POINTS

**Copyright permission**  
permissions@cacm.acm.org  
**Calendar items**  
calendar@cacm.acm.org  
**Change of address**  
acmhlp@cacm.acm.org  
**Letters to the Editor**  
letters@cacm.acm.org

**WEBSITE**  
<http://cacm.acm.org>

**AUTHOR GUIDELINES**  
<http://cacm.acm.org/>

#### ACM ADVERTISING DEPARTMENT

2 Penn Plaza, Suite 701, New York, NY  
10121-0701  
T (212) 626-0686  
F (212) 869-0481

**Director of Media Sales**  
Jennifer Ruzicka  
jen.ruzicka@hq.acm.org

**For display, corporate/brand advertising:**  
Craig Pitcher  
pitcherc@acm.org T (408) 778-0300  
William Sleight  
wsleight@acm.org T (408) 513-3408

**Media Kit** [acmm mediasales@acm.org](mailto:acmm mediasales@acm.org)

**Association for Computing Machinery (ACM)**  
2 Penn Plaza, Suite 701  
New York, NY 10121-0701 USA  
T (212) 869-7440; F (212) 869-0481

#### EDITORIAL BOARD

**EDITOR-IN-CHIEF**  
Moshe Y. Vardi  
eic@cacm.acm.org

#### NEWS

**Co-Chairs**  
William Pulleyblank and Marc Snir  
**Board Members**  
Mei Kobayashi; Michael Mitzenmacher;  
Rajeev Rastogi

#### VIEWPOINTS

**Co-Chairs**  
Tim Finin; Susanne E. Hambrusch;  
John Leslie King  
**Board Members**  
William Aspray; Stefan Bechtold;  
Michael L. Best; Judith Bishop;  
Stuart I. Feldman; Peter Freeman;  
Mark Guzdial; Rachelle Hollander;  
Richard Ladner; Carl Landwehr;  
Carlos Jose Pereira de Lucena;  
Beng Chin Ooi; Loren Terveen;  
Marshall Van Alstyne; Jeannette Wing

#### Q PRACTICE

**Co-Chair**  
Stephen Bourne  
**Board Members**  
Eric Allman; Peter Bailis; Terry Coatta;  
Stuart Feldman; Benjamin Fried;  
Pat Hanrahan; Tom Killalea; Tom Limoncelli;  
Kate Matsudaira; Marshall Kirk McKusick;  
George Neville-Niel; Theo Schlossnagle;  
Jim Waldo

The Practice section of the CACM Editorial Board also serves as the Editorial Board of *Queue*.

#### CONTRIBUTED ARTICLES

**Co-Chairs**  
Andrew Chien and James Larus  
**Board Members**  
William Aiello; Robert Austin; Elisa Bertino;  
Gilles Brassard; Kim Bruce; Alan Bundy;  
Peter Buneman; Peter Druschel; Carlo Ghezzi;  
Carl Gutwin; Yannis Ioannidis;  
Gal A. Kaminka; James Larus; Igor Markov;  
Gail C. Murphy; Bernhard Nebel;  
Lionel M. Ni; Kenton O'Hara; Sriram Rajamani;  
Marie-Christine Rousset; Avi Rubin;  
Krishan Sabnani; Ron Shamir; Yoav Shoham; Larry Snyder; Michael Vitale;  
Wolfgang Wahlster; Hannes Werthner;  
Reinhard Wilhelm

#### RESEARCH HIGHLIGHTS

**Co-Chairs**  
Azer Bestavros and Gregory Morrisett  
**Board Members**  
Martin Abadi; Amr El Abbadi; Sanjeev Arora;  
Nina Balcan; Dan Boneh; Andrei Broder;  
Doug Burger; Stuart K. Card; Jeff Chase;  
Jon Crowcroft; Sandhya Dwaekadas;  
Matt Dwyer; Alon Halevy; Norm Jouppi;  
Andrew B. Kahng; Sven Koenig; Xavier Leroy;  
Steve Marschner; Kobbi Nissim;  
Steve Seitz; Guy Steele, Jr.; David Wagner;  
Margaret H. Wright; Andreas Zeller

#### WEB

**Chair**  
James Landay  
**Board Members**  
Marti Hearst; Jason I. Hong;  
Jeff Johnson; Wendy E. MacKay

#### ACM Copyright Notice

Copyright © 2016 by Association for Computing Machinery, Inc. (ACM). Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and full citation on the first page. Copyright for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or fee. Request permission to publish from [permissions@acm.org](mailto:permissions@acm.org) or fax (212) 869-0481.

For other copying of articles that carry a code at the bottom of the first or last page or screen display, copying is permitted provided that the per-copy fee indicated in the code is paid through the Copyright Clearance Center; [www.copyright.com](http://www.copyright.com).

#### Subscriptions

An annual subscription cost is included in ACM member dues of \$99 (\$40 of which is allocated to a subscription to *Communications*); for students, cost is included in \$42 dues (\$20 of which is allocated to a *Communications* subscription). A nonmember annual subscription is \$269.

#### ACM Media Advertising Policy

*Communications of the ACM* and other ACM Media publications accept advertising in both print and electronic formats. All advertising in ACM Media publications is at the discretion of ACM and is intended to provide financial support for the various activities and services for ACM members. Current advertising rates can be found by visiting <http://www.acm-media.org> or by contacting ACM Media Sales at (212) 626-0686.

#### Single Copies

Single copies of *Communications of the ACM* are available for purchase. Please contact [acmhlp@acm.org](mailto:acmhlp@acm.org).

#### COMMUNICATIONS OF THE ACM

(ISSN 0001-0782) is published monthly by ACM Media, 2 Penn Plaza, Suite 701, New York, NY 10121-0701. Periodicals postage paid at New York, NY 10001, and other mailing offices.

#### POSTMASTER

Please send address changes to *Communications of the ACM*  
2 Penn Plaza, Suite 701  
New York, NY 10121-0701 USA

Printed in the U.S.A.



Association for  
Computing Machinery





DOI:10.1145/2945075

Moshe Y. Vardi

# The Ritual of Academic-Unit Review

**P**ERIODIC REVIEW OF academic units is a standard practice in academia. The goal of such review is to contribute to quality enhancement by providing the unit and the institution with a clear assessment of the unit's strengths and weaknesses. Typically, such a review consists of a thorough self-study by the reviewed unit, which is followed with a visit by a committee of experienced academicians. After the visit, the committee submits a report, which combines an incisive analysis with sage advice for unit improvements. Then what? Usually, not much, I am afraid. It is rare to see such a review resulting in a truly significant quality enhancement.

I recently had the opportunity to be a member of a committee that reviewed all 17 computer science programs in Israel. The 17 programs range from graduate programs at world-famous research institutions such as the Weizmann Institute of Science, to small undergraduate programs at small colleges such as Tel-Hai College at the Northern Galilee. The review project was carried out over a period of a year and a half, included three visits to Israel, and was, overall, a considerable effort. (The reports are available at [http://che.org.il/?page\\_id=34025](http://che.org.il/?page_id=34025).) Then what? Not much, I am afraid.

Why is it that academic-unit reviews accomplish so little in spite of the significant effort both by the reviewed units and reviewing committees? And why do we continue to conduct these reviews in spite of their meager results? The answer to the second question is clear, I believe. Unlike businesses, which can be measured by their bottom line, academic institutions have multiple, difficult to measure, and often conflicting goals. They are unified by the nebulous concept of academic

excellence. The pursuit of excellence must be visible, and academic-unit reviews serve this purpose. But why do they accomplish so little? There are three main reasons, I believe.

**Cultural Barriers:** Culture is defined as “the way of life, especially the general customs and beliefs, of a particular group of people at a particular time.” Each academic unit has its own unique culture. Culture usually creates social cohesion by means of shared expectations, but it can also be a barrier to change. Quality enhancement requires an academic unit to change the way it runs its business, but culture is persistent and change is difficult. Israel has become famous as the “Start-Up Nation,” with a thriving high-tech sector. The computer science programs in Israel create the educated workforce that underpins the Israel high-tech sector. Yet computer science units in Israel are generally spin-offs from mathematics units. As a result they tend to be highly theoretical in their research focus. The Review Committee's report called for a better balance between theoretical and experimental research, but such a change runs against an entrenched culture.

**Institutional Barriers:** Academic-unit reviews typically focus on low-level units, such as departments. But departments do not operate in a vacuum. Departments typically belong to schools, and the operations of a department must be considered in the context of the school in which it is housed. But reviews almost always focus narrowly on departmental operations, missing the bigger context. Furthermore, academic units have multiple stakeholders—academic staff members, administrative staff members, students, and higher-level academic administrators—with di-

verse sets of interests. Almost every proposed quality-enhancing change runs against the interests of some stakeholder. Thus, there are institutional barriers to change. It takes very strong institutional support for quality enhancement—as well as committed leadership at all levels—to implement recommended changes. For example, in one institution in Israel we found five distinct computer science programs and called for an institutional reorganization to increase synergy and efficiency. But such reorganization runs against several institutional interests and is unlikely to happen.

**Follow-Up Process:** The reviews produced by review committees often offer insightful diagnoses of the weaknesses discovered. They usually also offer recommendations for improvements, but they usually do not offer detailed action plans, as reviewers lack the intimate institutional knowledge required to develop such plans. But most institutions do not have a robust follow-up process to ensure that a detailed action plan to deal with the recommendations is developed by department and the school, is carried out to completion, and then re-reviewed to ensure actual improvement. Thus, academic-unit review reports mostly gather dust rather than affect change.

So what is to be done? Should we abandon the ritual of academic-unit review? I view the pursuit of academic excellence as a hallowed academic value, and reviews can serve an important role in such a pursuit, but they should not be undertaken without utmost commitment to the process and fortitude to carry it out.

Follow me on Facebook, Google+, and Twitter.

**Moshe Y. Vardi**, EDITOR-IN-CHIEF

Copyright held by author.

# SHAPE THE FUTURE OF COMPUTING.

## JOIN ACM TODAY.

ACM is the world's largest computing society, offering benefits and resources that can advance your career and enrich your knowledge. We dare to be the best we can be, believing what we do is a force for good, and in joining together to shape the future of computing.

### SELECT ONE MEMBERSHIP OPTION

#### ACM PROFESSIONAL MEMBERSHIP:

- ☐ Professional Membership: \$99 USD
- ☐ Professional Membership plus  
ACM Digital Library: \$198 USD (\$99 dues + \$99 DL)
- ☐ ACM Digital Library: \$99 USD  
(must be an ACM member)

#### ACM STUDENT MEMBERSHIP:

- ☐ Student Membership: \$19 USD
- ☐ Student Membership plus ACM Digital Library: \$42 USD
- ☐ Student Membership plus Print *CACM* Magazine: \$42 USD
- ☐ Student Membership with ACM Digital Library plus  
Print *CACM* Magazine: \$62 USD

- ☐ **Join ACM-W:** ACM-W supports, celebrates, and advocates internationally for the full engagement of women in all aspects of the computing field. Available at no additional cost.

Priority Code: CAPP

#### Payment Information

Name \_\_\_\_\_

ACM Member # \_\_\_\_\_

Mailing Address \_\_\_\_\_

City/State/Province \_\_\_\_\_

ZIP/Postal Code/Country \_\_\_\_\_

Email \_\_\_\_\_

Payment must accompany application. If paying by check or money order, make payable to ACM, Inc., in U.S. dollars or equivalent in foreign currency.

☐ AMEX ☐ VISA/MasterCard ☐ Check/money order

Total Amount Due \_\_\_\_\_

Credit Card # \_\_\_\_\_

Exp. Date \_\_\_\_\_

Signature \_\_\_\_\_

#### Purposes of ACM

ACM is dedicated to:

- 1) Advancing the art, science, engineering, and application of information technology
- 2) Fostering the open interchange of information to serve both professionals and the public
- 3) Promoting the highest professional and ethics standards

Return completed application to:  
ACM General Post Office  
P.O. Box 30777  
New York, NY 10087-0777

Prices include surface delivery charge. Expedited Air Service, which is a partial air freight delivery service, is available outside North America. Contact ACM for more information.

**Satisfaction Guaranteed!**

## BE CREATIVE. STAY CONNECTED. KEEP INVENTING.



Association for  
Computing Machinery

1-800-342-6626 (US & Canada)  
1-212-626-0500 (Global)

Hours: 8:30AM - 4:30PM (US EST)  
Fax: 212-944-1318

acmhelp@acm.org  
acm.org/join/CAPP



DOI:10.1145/2949336

Vinton G. Cerf

# The Power of Big Ideas

I've been thinking about "big ideas" recently and how powerful they can be. What strikes me is that so many of these ideas are so easily stated, such as "Cure Cancer" or "Put a man

on the moon," but are really difficult to accomplish. Their simplicity hides enormous challenges, infinite details. At Google, the company began with a goal "to organize the world's information and make it universally accessible and useful." As simply as this goal can be stated and, in some sense, understood, it also is a driver for an endless array of initiatives, intermediate goals, blind alleys, successes, and failures. One can adhere to this goal while motivating the development of machine translation, indexing and rank ordering searches of the World Wide Web, evolving massive datacenters and high capacity fiber networks and collaborative applications that allow concurrent editing of documents, presentations, and spreadsheets, to mention just a few examples.

One reason these easily stated big ideas are so powerful is that one can test whatever one is doing against the idea to see whether the work contributes to achieving the goal. If your goal is to make autonomous, self-driving vehicles, it's easy to ask: "Is what I am doing contributing to this goal and if so, how?" This facet also helps multiple groups appreciate the work of others, assuming their work is understood to be relevant to the objective in hand. Big ideas are, in some ways, organizing principles that point in the direction of useful research, development, and engineering. Interestingly, these simply stated goals may also motivate the development of business models for sus-

tainability, where the goal implies the need for continued, long-term maintenance and operation.

The Internet started with a simply stated goal of connecting any number and kind of packet switched networks. As complex as the Internet has become, this is still a driving factor in its evolution. The arrival of smartphones and high-speed, digital wireless communication, Wi-Fi, optical fiber networks, and many other networking technologies have been integrated into the global network of networks we have come to view as the Internet.

Artificial intelligence has been a long-sought goal in the computer science field, speculated about since the earliest days of computing. Alan Turing contemplated about this in 1950 with his famous paper on "Computing Machinery and Intelligence"<sup>a</sup> and Vannevar Bush hinted at this in his equally if not more famous paper "As We May Think."<sup>b</sup> Bush draws us into a vision that echoed and advanced with the work of J.C.R. Licklider and Douglas Engelbart, two giants in the fields of information processing and human/computer partnerships. With the notable successes of IBM's Deep Blue,<sup>c</sup> IBM's Watson,<sup>d</sup> and Google/DeepMind's AlphaGo,<sup>e</sup> we can see examples of powerful specialization at

work. DeepMind's work appears to be generalizable to a number of board games and the multilayer neural networks from which DeepMind is fashioned seem able to learn how to play without specific programming. What is needed is exposure to many, many games and feedback as to success or failure. Whether such methods can be applied to what is called General Problem Solving isn't entirely clear to me. The latter involves modeling, goal setting, hypothesis generation, experimental exploration, and testing—all of which remind me in some ways of the general scientific method.

As we explore the potential partnerships and collaborations between computing systems and their human users, many of us hope that we will accelerate progress in science, technology, applications, and economic conditions. Already we can see evidence that robotic factories are far more flexible than earlier special-purpose manufacturing systems. 3D printing, coupled with CAD/CAM and machine-aided design, may take us further along the path to what has sometimes been called "mass customization." If productivity increases, costs go down, making more things affordable. Jobs may well disappear but they will be replaced with new jobs. The conundrum is whether those displaced from older jobs can be equipped to carry out the new ones that are created. The big idea that comes to my mind is: "improve the well-being of every person on the planet."

Note in passing: A remarkable milestone for women filling all the leadership roles at ACM: President, Vice-President and Secretary/Treasurer. At the National Science Board a similar outcome: Chair, Vice-Chair, and Director of the National Science Foundation. 

**Vinton G. Cerf** is vice president and Chief Internet Evangelist at Google. He served as ACM president from 2012–2014.

Copyright held by author.

a <http://www.loebner.net/Prize/TuringArticle.html>

b <http://www.theatlantic.com/magazine/archive/1945/07/as-we-may-think/303881/>

c Deep Blue beat Garry Kasparov at chess in 1997.

d Watson wins "Jeopardy!" in 2011.

e DeepMind's AlphaGo wins at Go in 2016.

## Results of ACM's 2016 General Election

### President:

Vicki L. Hanson  
(term: July 1, 2016 – June 30, 2018)

### Vice President:

Cherri Pancake  
(term: July 1, 2016 – June 30, 2018)

### Secretary/Treasurer:

Elizabeth Churchill  
(term: July 1, 2016 – June 30, 2018)

### Members at Large:

Gabriele Anderst-Kotsis  
(term: July 1, 2016 – June 30, 2020)

Susan Dumais  
(term: July 1, 2016 – June 30, 2020)

Elizabeth Mynatt  
(term: July 1, 2016 – June 30, 2020)

Pam Samuelson  
(term: July 1, 2016 – June 30, 2020)

Eugene H. Spafford  
(term: July 1, 2016 – June 30, 2020)



# Rethinking Computational Thinking

**H**OW IMPORTANT ARE skills in computational thinking for computing app constructors and for computing users in general? If we can teach our children early on to smile, talk, write, read, and count through frequent and repetitive use of patterns in well-chosen examples, is it also possible for us, assuming we have the skills, to teach our children to construct computing applications? Do we need to first teach them anything about computational thinking before we look to teach how to construct computing apps? If not, how important will computational skills be for us all, as Jeannette Wing suggests in her blog@cacm “Computational Thinking, 10 Years Later” (Mar. 2016 and reprinted on p. 10) and Viewpoint column “Computational Thinking” (Mar. 2006)?

Many competent and successful computing app constructors and users never hear a word about computational thinking but still manage to acquire sufficient construction and user skills through frequent and repetitive use of patterns in well-chosen examples. Can computing app constructors learn their skills from frequent and repetitive use of the patterns in well-chosen examples without computational thinking? Current results, as reflected in app stores, seem to show they can.

What specific skills define computational thinking? Do constructors learn them only through frequent and repetitive use of patterns in well-chosen examples? Time will tell. Does this mean skills in computational thinking are of little or no value? Not at all. But it does mean they need to be learned the same way skills in smiling, talking, writing, reading, and counting are learned—at the appropriate time.

What if experts would first have to demonstrate their own practical skills in computational thinking before they could promote the theoretical virtues

of such disciplines? Would it avoid the dangers of the less computational savvy among us being led astray by well-intentioned fervor?

What if Jeannette Wing would publish the steps she uses to identify, specify, design, and code, say, a phone or tablet app to display an album of photos and supporting text? Who would benefit most? Would it be only Microsoft employees or many more of us constructors and users out in the world? Would we even learn the skills of computational thinking?

All practice and no theory? No, practice *and* theory. But practice first please, then theory.

**Fred Botha**, Sun City West, AZ

### Teaching the Art of Delegation

I commend Kate Matsudaira’s wonderful article “Delegation as Art” (May 2016), with its spot-on guidance for addressing challenges in mentoring and delegating our co-workers and students; like Matsudaira, I love to imagine my teammates asking themselves, “What would Geoff say?” Matsudaira also did a superb job explaining her suggestions in ways that make them applicable to disciplines well beyond engineering.

However, one key challenge in managing software engineers Matsudaira did not address is that senior engineers are often expected to mentor team members while simultaneously being responsible for delivering the very projects on which the mentees are working. Matsudaira did say mentoring and delegation require letting people find their own way and make mistakes, even as project success is often measured by speed of delivery and perfection. As a result, mentoring success sometimes comes at the expense of project success, and vice versa.

It would benefit us all if our managers and project managers had a better understanding of the value and process of mentoring. To this

end, I will be sharing the article with fellow leaders in my organization and recommend you share it with yours as well.

**Geoffrey A. Lowney**, Issaquah, WA

### Professionalize CS Education Now

Computer science as a discipline has been remarkably bad at marketing itself. With current record enrollments in American universities, this is not a problem, but consider how it might play out in an uncertain future. People today using computers, cellphones, tablets, and automated highway toll payment devices, as well as multiple websites and services on a daily basis lack a clear idea what computer scientists actually do, and that they are indeed professionals, like lawyers, accountants, and medical doctors.

Parents concerned about the earning potential of their children and of their children's future spouses forever try to address the conundrum of what academic path to take, as in, say, medical school, law school, or business school.

I thus propose a simple semantic change, and the far-reaching organizational change it implies. Master's programs in computer science shall henceforth graduate "computists" and rebrand themselves as "computing schools." Future parents and parents-in-law should be able to choose among, say, medical school, computing school, business school, and law school. Existing Master's curricula in computer science would be extended to five or six semesters, with mandatory courses in all areas of applied computing, from the perspectives of building systems and selecting, evaluating, adapting, and applying existing systems in all facets of computing, based on scientific methods and precise metrics.

When this basic framework is established, computing schools would stop accepting the Graduate Records Exam (<http://www.ets.org/gre>) and switch to their own dedicated entrance exam, in the same way the LSAT is used for law school and the MCAT for medical school. The conferred degree would be called, say, Professional Master of Computing.

It might take a decade or more to

change the perception of the American public, but such change is essential and should be embraced as quickly as possible and on a national basis. ACM is best positioned and able to provide the leadership needed to move this important step forward for the overall discipline of computer science.

**James Geller**, Newark, NJ

### Causality in Computer Science

I would like to congratulate Carlos Baquero and Nuno Preguiça for their clear writing and the good examples they included in their article "Why Logical Clocks Are Easy" (Apr. 2016), especially on a subject that is not easily explained. I should say the subject of the article is quite far from my usual area of research, which is, today, formal methods in security. Still, we should reflect on Baquero's and Preguiça's extensive use of the concept of "causality." That concept has been used in science since ancient Greece, where it was developed by the atomists, then further, to a great extent, by Aristotle, through whose writings it reached the modern world.

The concept of causality was criticized by David Hume in the 18<sup>th</sup> century. Commenting on Hume, Bertrand Russell (in his 1945 book *A History of Western Philosophy*) said, "It appears that simple rules of the form 'A causes B' are never to be admitted in science, except as crude suggestions in early stages." Much of modern science is built on powerful equations from which many causal relationships can be derived, the implication being the latter are only explanations or illustrations for the relationships expressed by the former.

Causal laws are not used in several well-developed areas of computer science, notably complexity theory and formal semantics. In them, researchers write equations or other mathematical or logical expressions. At one point in their article, Baquero and Preguiça redefined causality in terms of set inclusion. Leslie Lamport's classic 1978 paper "Time Clocks, and the Ordering of Events in a Distributed System" (cited by Baquero and Preguiça) seems to use the concept of causality for explanations, rather than for the

core theory. In several papers, Joseph Y. Halpern and Judea Pearl have developed the concept of causality in artificial intelligence, but their motivations and examples suggest application to "early stage" research, just as Bertrand Russell wrote.

I submitted this letter mainly to prompt thinking on what role the causality concept should play in the progress of various areas of computer science. Today, it is used in computer systems, software engineering, and artificial intelligence, among others, probably. Should we thus aim for its progressive elimination, leaving it a role only in the area of explanations and early-stage intuitions?

**Luigi Logrippo**, Ottawa-Gatineau, Canada

*Communications* welcomes your opinion. To submit a Letter to the Editor, please limit yourself to 500 words or less, and send to [letters@cacm.acm.org](mailto:letters@cacm.acm.org).

© 2016 ACM 0001-0782/16/07 \$15.00

Coming Next Month in COMMUNICATIONS

**Computational Biology  
in the 21<sup>st</sup> Century**

**Smart Cities**

**Debugging  
Distributed Systems**

**Skills for Success Across  
Three IT Career Stages**

**Adaptive Computation: In  
Memory of John Holland**

**Ur/Web: A Simple Model  
for Programming the Web**

**Verifying Quantitative  
Reliability for Programs  
that Execute on  
Unreliable Hardware**

Plus the latest news about deep reinforcement learning, the value of open source, and apps for social good.

The *Communications* Web site, <http://cacm.acm.org>, features more than a dozen bloggers in the BLOG@CACM community. In each issue of *Communications*, we'll publish selected posts or excerpts.



Follow us on Twitter at <http://twitter.com/blogCACM>

DOI:10.1145/2933410

<http://cacm.acm.org/blogs/blog-cacm>

## Progress in Computational Thinking, and Expanding the HPC Community

*Jeannette Wing considers the proliferation of computational thinking, while Dan Stanzione hopes to bring more HPC practitioners to SC16.*



**Jeannette Wing**  
**Computational Thinking, 10 Years Later**  
<http://bit.ly/1WAXka7>  
March 23, 2016

“Not in my lifetime.”

That is what I said when I was asked whether we would ever see computer science taught in K–12. It was 2009, and I was addressing a gathering of attendees to a workshop on computational thinking (<http://bit.ly/1NjmcRJ>) convened by the National Academies.

I am happy to say that I was wrong.

It has been 10 years since I published my three-page “Computational Thinking” Viewpoint (<http://bit.ly/1W73ekv>) in the March 2006 issue of *Communications*. To celebrate its anniversary, let us consider how far we’ve come.

Think back to 2005. Since the dot-com bust, there had been a steep and steady decline in undergraduate enrollments in computer science, with

no end in sight. The computer science community was wringing its hands, worried about the survival of its departments on campuses. Unlike many of my colleagues, I saw a different, much rosier future for computer science. I saw computing was going to be everywhere.

I argued the use of computational concepts, methods, and tools would transform the very conduct of every discipline, profession, and sector. Someone with the ability to use computation effectively would have an edge over someone without. So, I saw a great opportunity for the computer science community to teach future generations how computer scientists think. Hence, “computational thinking.”

I must admit, I am surprised and gratified by how much progress we have made in achieving this vision: Computational thinking will be a fundamental skill used by everyone in the world by the middle of the 21st century. By fundamental, I mean as fundamental as reading, writing, and arithmetic.

### The Third Pillar

I knew in the science and engineering disciplines, computation would be the third pillar of the scientific method, along with theory and experimentation. After all, computers were already used for simulation of large, complex, physical and natural systems. Sooner or later, scientists and engineers of all kinds would recognize the power of computational abstractions, such as algorithms, data types, and state machines.

Today, with the advent of massive amounts of data, researchers in all disciplines—including the arts, humanities and social sciences—are discovering new knowledge using computational methods and tools.

In the past 10 years, I visited nearly 100 colleges and universities worldwide and witnessed a transformation at the undergraduate level. Computer science courses are now offered to students not majoring in computer science. These courses are not computer programming courses, but rather focus on core concepts in computer science. At Harvard, this course (CS50) is one of the most popular courses (<http://bit.ly/1SZLuqe>), not just on its campus but also at rival Yale’s campus. And what about computer science enrollments? They are skyrocketing (<http://bit.ly/1Tt909p>)!

Perhaps the most surprising and gratifying result is what is happening at the K–12 level. First, the U.K.’s grassroots effort Computing At School (<http://www.computingatschool.org.uk/>) led the Department of Education to require computing in K–12 schools in England starting September 2014. The statutory

guidance for the national curriculum says, “A high-quality computing education equips pupils to use computational thinking and creativity to understand and change the world.”

In addition, the BBC in partnership with Microsoft and other companies funded the design and distribution of the BBC micro:bit (<https://www.micro-bit.co.uk/>). One million of these programmable devices were distributed free earlier this year (March 2016), one for every 11–12-year-old (Year 7) student in the U.K., along with their teachers. Microsoft Research contributed to the design and testing of the device, and the MSR Labs Touch Develop team provided a programming language and platform for the BBC micro:bit, as well as teaching materials.

Second, code.org is a nonprofit organization, started in 2013, dedicated to the mission of providing access to computer science education to all. Microsoft, along with hundreds of other corporate and organizational partners, helps sponsor the activities of code.org.

Third, internationally there is a groundswell of interest in teaching computer science at the K–12 level. I know of efforts in Australia, Israel, Singapore, and South Korea. China is likely to make a push soon, too.

### Computer Science for All

Most gratifying to me is President Barack Obama’s pledge to provide \$4 billion in funding for computer science education in U.S. schools as part of the Computer Science for All Initiative (<http://1.usa.gov/21u4mxK>) he announced on Jan. 30. That initiative includes \$120 million from the National Science Foundation, which will be used to train as many as 9,000 more high school teachers to teach computer science and integrate computational thinking into their curriculum. This push for all students to learn computer science comes partly from market demand for workers skilled in computing from all sectors, not just information technology. We see this at Microsoft, too; our enterprise customers in all sectors are coming to Microsoft because they need more computing expertise.

Practical challenges and research opportunities remain. The main practical challenge is that we do not have enough K–12 teachers trained to teach comput-

er science to K–12 students. I am optimistic that, over time, we will solve this.

There also are interesting research questions I would encourage computer scientists to pursue, working with the cognitive and learning sciences communities. First, what computer science concepts should be taught when, and how?

Consider an analogy to mathematics. We teach numbers to 5-year-olds, algebra to 12-year-olds, and calculus to 18-year-olds. We have somehow figured out the progression of concepts to teach in mathematics, where learning one new concept builds on understanding the previous concept, and where the progression reflects the progression of mathematical sophistication of a child as he or she matures.

What is that progression in computer science? For example, when is it best to teach recursion? Children learn to solve the Towers of Hanoi puzzle (for small  $n$ ), and in history class we teach “divide and conquer” as a strategy for winning battles. But is the general concept better taught in high school? We teach long division to 9-year-olds in 4th grade, but we never utter the word “algorithm.” And yet the way it is taught, long division is just an algorithm. Is teaching the general concept of an algorithm too soon for a 4th grader? More deeply, are there concepts in computing that are innate and do not need to be formally learned?

Second, we need to understand how best to use computing technology in the classroom. Throwing computers in the classroom is not the most effective way to teach computer science concepts. How can we use technology to enhance the learning and reinforce the understanding of computer science concepts? How can we use technology to measure progress, learning outcomes, and retention over time? How can we use technology to personalize the learning for individual learners, as each of us learns at a different pace and has different cognitive abilities?

We have made tremendous progress in injecting computational thinking into research and education of all fields in the last 10 years. We still have a ways to go, but fortunately, academia, industry and government forces are aligned toward realizing the vision of making computational thinking commonplace.



**Dan Stanzione**  
**SC16 Expands Focus**  
**on HPC Provider**  
**Community,**  
**Practitioners**  
<http://bit.ly/1REKjKI>  
 April 6, 2016

If you are in HPC (high-performance computing) or a related field, you know SC16 (<http://sc16.supercomputing.org/>) as the leading international conference for high-performance computing, networking, storage, and analysis. For 28 years, SC has served as the conference of record in the supercomputing community for presenting the results of groundbreaking new research, getting the training needed to advance your career, and discovering what is new in the marketplace.

SC16 marks the beginning of a multi-year emphasis designed to advance the state of the practice in the HPC community by providing a track for professionals driving innovation and development in designing, building, and operating the world’s largest supercomputers, along with the system and application software that make them run effectively. We call this the “State of the Practice.”

“State of the Practice” submissions will add content about innovations and best practices development from the HPC service provider community into all aspects of the SC16 technical program (<http://bit.ly/1TOZ6yx>), from tutorials and workshops to papers and posters. These submissions will be peer reviewed, as are all submissions to SC. However, the evaluation criteria will acknowledge the fundamental difference between innovation that leads the state of HPC practice in the field today, and research results that will reshape the field tomorrow.

If you are part of the SC community but have not always felt SC was the right venue to showcase your contributions to the HPC body of knowledge, we want to encourage you to submit to the technical program on the reinvigorated “State of the Practice” track.

Check the important dates page (<http://bit.ly/1WaLX9j>) for upcoming submission deadlines. 

**Jeannette M. Wing** is corporate vice president at Microsoft Research. **Dan Stanzione** is Executive Director of the Texas Advanced Computing Center at the University of Texas at Austin and serves as co-chair of the “State of the Practice” track at SC16.

© 2016 ACM 0001-0782/16/07 \$15.00

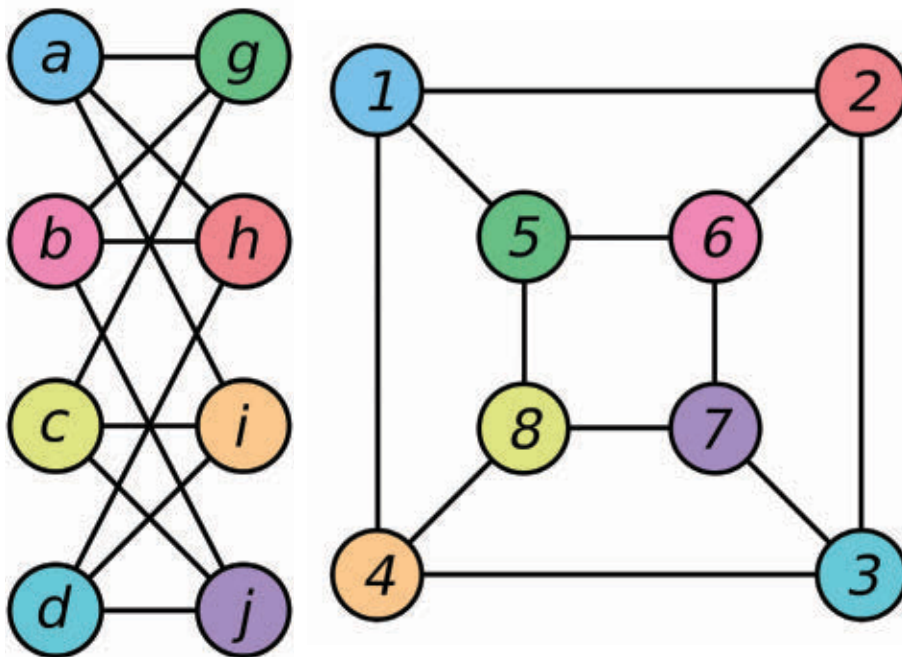
# Graph Matching in Theory and Practice

*A theoretical breakthrough in graph isomorphism excites complexity experts, but will it lead to any practical improvements?*

**B**ACK IN 1979, two scientists wrote a seminal textbook on computational complexity theory, describing how some problems are hard to solve. The known algorithms for handling them grow in complexity so fast that no computer can be guaranteed to solve even moderately sized problems in the lifetime of the universe. While most problems could be deemed either relatively easy or hard for a computer to solve, a few fell into a strange nether region where they could not be classified as either. The authors, Michael Garey and David S. Johnson, helpfully provided an appendix listing a dozen problems not known to fit into one category or the other.

“The very first one that’s listed is graph isomorphism,” says Lance Fortnow, chair of computer science at the Georgia Institute of Technology. In the decades since, most of the problems on that list were slotted into one of the two categories, but solving graph isomorphism—in essence figuring out if two graphs that look different are in fact the same—resisted categorization. “Graph isomorphism just didn’t fall.”

Now László Babai, a professor of computer science and mathematics



Two graphs that are isomorphic.

at the University of Chicago, has developed a new algorithm for the problem that pushes it much closer to—but not all the way into—the easy category, a result complexity experts are hailing as a major theoretical achievement, although whether his work will have any practical effect is unclear.

In computer science, a graph is a network, a series of nodes strung together by connections known as edges; the set of Facebook users and their interconnections make up a graph with approximately 1.5 billion nodes. Two graphs may be depicted differently, but if the nodes and connections in

one match up with those in the other, they are equivalent—that is, isomorphic—graphs. Graph isomorphism can be used for applications as diverse as searching chemical databases or performing facial recognition; is that the same molecule or the same face, even though the atoms have been flipped or the smile has become a grimace? “Once you know two things are the same, then you can take what you know about one and apply it to the other,” Fortnow says.

Graph isomorphism is a subset of string isomorphism; in fact, string isomorphism is the subject of Babai’s paper, although it mainly discusses graphs. While there had been solutions for various special cases of graphs, “this works for everything,” says Fortnow, who describes Babai as having been his mentor when he started his career as an assistant professor at the University of Chicago.

What Babai’s solution does is check subsections of graphs for isomorphism through a variety of relatively simple means; for instance, by comparing how many neighbors each node has. Trouble arises if any of those subsections turn out to be Johnson graphs, which are so highly symmetrical, it can be difficult to tell if one is the same as another.

“Basically, what it boils down to is he shows that if this process gets stuck, there must be a Johnson graph hiding there,” Fortnow says. Having uncovered a hidden Johnson graph, Babai’s paper describes how to check that graph.

Babai first presented his results in a talk in November 2015, and in December published a paper on the open access website ArXiv. The paper has been accepted to the 48th ACM Symposium on Theory of Computing (STOC 2016). Babai declined an interview, saying the work needed to be vetted by other computer scientists. Fortnow says that, while it is a good idea to have many people look over such a complicated proof, the results are probably sound. “I think the rest of us really believe it’s either correct as is, or if there’s some problem, it can be easily fixed,” he says.

## P or NP?

The difficulty with graph isomorphism has been figuring out what kind of problem it is. Many problems are considered “easy.” As they grow larger, the time it takes a computer to solve them

## What Babai’s solution does is check subsections of graphs for isomorphism through a variety of relatively simple means.

increases no faster than a polynomial function of the problem’s size. The time needed to solve these P problems can grow large, but not unreasonably so. At the other end of the scale are problems in which the solution time of any known algorithm probably grows exponentially with the size of the problem, to the point where it becomes unfeasible for the computer.

The P problems that can be solved in polynomial time belong to a larger class known as NP. NP includes any problem in which, regardless of how difficult it is to solve, showing the solution is true is easy. The Traveling Salesman problem is a classic example; it asks for the shortest route that allows a traveling salesman to visit a list of cities once and return to the starting point. Coming up with that route is challenging, but once given the route, it can be easily checked.

Another subset of NP is NP-complete. No polynomial algorithm is known for any NP-complete problem, but if one were discovered, there would be an algorithm for all of them and the two classes, P and NP-complete, would collapse into one. Scientists do not know whether no efficient algorithm exists for solving NP-complete problems, or whether they have simply failed to find one yet. Since 2000, the Clay Mathematics Institute has offered a \$1 million prize to anyone who can show whether P equals NP.

Babai’s work does not answer that question, but it still intrigues theoreticians. Graph isomorphism has never been shown to be in P, because there is as yet no known algorithm to solve it in polynomial time. On the other hand,

# ACM Member News

## INDYK STRIVES TO BOOST EFFICIENCY OF ALGORITHMS



“My research interests are generally in the design and analysis of efficient algorithms,”

says Piotr Indyk of the Theory of Computation Group of the Massachusetts Institute of Technology (MIT) Computer Science and Artificial Intelligence Laboratory.

“I am interested in developing solutions to fundamental algorithmic problems that impact the practice of computing,” adds Indyk. “Some examples include high-dimensional similarity search, which involves finding pairs of high-dimensional vectors that are close to each other; faster Fourier transform algorithms for signals with sparse spectra, and so forth.”

Born in Poland, Indyk received a Magister degree from the University of Warsaw in 1995, and a Ph.D. in computer science from Stanford University in 2000. That same year, he joined the faculty of MIT, where he has been ever since.

“Recently, I managed to make some progress on whether we can show, or at least provide evidence, that some of those problems cannot be solved faster than what is currently known,” Indyk said. “I have also been working on proving there are certain natural problems—like regular expression matching, for example—that require quadratic time to solve, assuming some natural conjectures or hypothesis. This has been my main focus over the past year, which is a little unusual for me. This is because I typically work on algorithms and try to make them faster, as opposed to showing they might not be improved.”

During an upcoming sabbatical at Stanford University, Indyk plans to work with researchers who develop tools for proving conditional hardness of certain problems. “Stanford is my alma mater, so I am looking forward to it.”

—John Delaney

there is evidence it is not NP-complete either. The best theoretical algorithm, published by Babai and Eugene Luks in 1983, was sub-exponential—better than any known algorithm for any NP-complete problem, but still far from easy. This new algorithm appears to place graph isomorphism in quasipolynomial time, much closer to the easy side. “This is just such a huge theoretical improvement from what we had before,” Fortnow says. “Now it’s not likely at all to be NP-complete and it’s not that hard at all.”

Josh Grochow, an Omidyar Postdoctoral Fellow in theoretical computer science at the Santa Fe Institute in New Mexico, calls Babai’s proposed solution a big theoretical jump. “There really are very few problems we know of in this limbo state,” he says. Graph isomorphism is “still in the limbo state, but it’s a lot clearer where it sits.”

In some subset of graphs, the question was already settled. For instance, molecular graphs have “bounded valence,” meaning the physical constraints of three-dimensional space allow atoms to be connected only to a limited number of other atoms, says Jean-Loup Faulon, a synthetic biologist at the University of Manchester, England, and INRA in France, who uses graph matching in his work. In the 1980s, he points out, Babai, Luks, and Christoph Hoffmann showed bounded valence graphs could be solved in polynomial time. “Therefore, the problem of graph isomorphism is already known to be polynomial for chemicals,” Faulon says.

While Babai’s work has generated much excitement among theoreticians, experts say it is not likely to have much effect in practice; at least, not immediately. Scott Aaronson, a computer scientist at the Massachusetts Institute of Technology who studies computational theory, says, “this is obviously the greatest theoretical algorithms result” at least since 2002, when Manindra Agrawal, Neeraj Kayal, and Nitin Saxena came up with an algorithm for determining, in polynomial time, whether a number is a prime. “On the other hand,” Aaronson says, “the immediate *practical* impact on computing is zero, because we already had graph isomorphism algorithms that were extremely fast in practice for any

graphs anyone would ever care about.”

In fact, the graph isomorphism solution is similar to the Agrawal-Kayal-Saxena algorithm in that way, says Boaz Barak, Gordon McKay professor of Computer Science at Harvard University’s John A. Paulson School of Engineering and Applied Sciences. “It was a great intellectual breakthrough, but even before it, we had highly efficient heuristics that seemed to solve the problem.” Barak says Babai’s paper contains several significant new ideas that may have an effect on mathematics and may someday lead to practical applications in graph isomorphism or some other problem, adding, “however, it is impossible, at least for me, to predict this potential impact at this point.”

### Separate Spheres

Practical solutions for specific cases of graph isomorphism have existed for years. In a 1981 paper entitled “Practical Graph Isomorphism,” computer scientist Brendan McKay of the Australian National University in Canberra developed an isomorphism testing program called nauty (for No AUTomorphisms, Yes?) that is widely used today. He describes the difference between the theory and practice of graph isomorphism “as being like two galaxies that have only a few stars in common.” McKay does not believe this breakthrough will help close the gap between theory and practice, a belief he says he shares with Babai. “Nobody knows how to apply any of Babai’s ideas to improve the practical performance of programs for graph isomorphism,” McKay says. “If one just programs Babai’s algorithm on a computer without making any changes, it will be hopelessly slow compared to the best current programs.”

Faulon hopes Babai’s ideas do eventually lead to some changes in practice. “We are still lacking a practical polynomial-time algorithm in chemistry,” he says.

Another person hoping the theory can be translated into practice is Hasan Jamil, a computer scientist at the University of Idaho’s Institute for Bioinformatics and Evolutionary Studies. Jamil is searching for a gene that affects the death of cells in neurological disorders such as Alzheimer’s and

Parkinson’s diseases. Graph matching for the cells’ regulatory networks and protein networks could help him isolate such a gene or set of genes. When he submitted a paper and the reviewers asked for a new experiment to back up his theory, however, the publication process slowed to a crawl. “If I have to find an isomorphic match, or even an approximate match, it takes hours and hours and months to even find simple things,” he says. With approximately 24,000 genes in the human body, each of them capable of producing four or five different proteins and then of interacting amongst themselves, the number of possible matches quickly becomes astronomical.

He hopes Babai’s proposed solution leads to something better. “If it brings down the complexity, even a bit, it helps us search things in a reasonable amount of time,” Jamil says.

If Babai’s work does lead to practical improvements, they may lie far in the future. McKay says, though, that Babai’s ideas have started him thinking about whether there might be some useful application. “The odds don’t look good,” McKay says, “but just maybe there is a pleasant surprise lurking.” 

### Further Reading

Babai, L.  
Graph Isomorphism in Quasipolynomial Time, arXiv, Cornell University Library, 2015, <http://arxiv.org/abs/1512.03547>

McKay, B.D., and Piperno, A.  
Practical Graph Isomorphism II, *Journal of Symbolic Computation*, 60, 2014, <http://arxiv.org/abs/1301.1493>

Fortnow, L.  
The status of the P versus NP problem, *Communications of the ACM*, September 2009, <http://cacm.acm.org/magazines/2009/9/38904-the-status-of-the-p-versus-np-problem/fulltext>

Fortnow, L., and Grochow, J.A.  
Complexity classes of equivalence problems revisited, *Information and Computation*, 209, 2011, <http://arxiv.org/pdf/0907.4775.pdf>

Isomorphic Graphs, Example 1, Stats-Lab Dublin  
<https://www.youtube.com/watch?v=Xq8o-z1DsUA>

Neil Savage is a science and technology writer based in Lowell, MA.

© 2016 ACM 0001-0782/16/07 \$15.00

# Accelerating Search

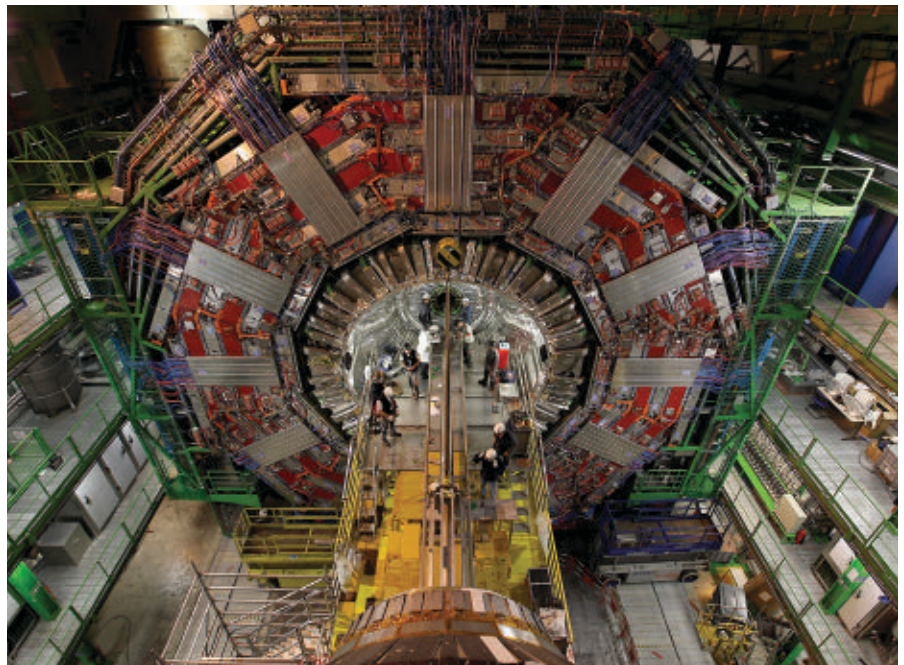
*The latest in machine learning helps high-energy physicists handle the enormous amounts of data produced by the Large Hadron Collider.*

EVERYTHING ABOUT THE Large Hadron Collider (LHC), the particle accelerator most famous for the Nobel Prize-winning discovery of the elusive Higgs boson, is massive—from its sheer size to the grandeur of its ambition to unlock some of the most fundamental secrets of the universe. At 27 kilometers (17 miles) in circumference, the accelerator is easily the largest machine in the world. This size enables the LHC, housed deep beneath the ground at CERN (the European Organization for Nuclear Research) near Geneva, to accelerate protons to speeds infinitesimally close to the speed of light, thus creating proton-on-proton collisions powerful enough to recreate miniature Big Bangs.

The data about the output of these collisions, which is processed and analyzed by a worldwide network of computing centers and thousands of scientists, is measured in petabytes: for example, one of the LHC's main pixel detectors, the ultra-durable high-precision cameras that capture information about these collisions, records an astounding 40 million pictures *per second*—far too much to store in its entirety.

This is the epitome of big data—yet when we think of big data, and of the machine-learning algorithms used to make sense of it, we usually think of applications in text processing and computer vision, and of uses in marketing by the likes of Google, Facebook, Apple, and Amazon. “The center of mass in applications is elsewhere,” outside of the physical and natural sciences, says Isabelle Guyon of the University of Paris-Saclay, who is the university's chaired professor in big data. “So even though physics and chemistry are very important applications, they don't get as much attention from the machine learning community.”

Guyon, who is also president of ChaLearn.org, a non-profit that organizes machine-learning competi-



Workers insert a new CMS Beam Pipe during maintenance on the Large Hadron Collider.

tions, has worked to shift data scientists' attention toward the needs of high-energy physics. The Higgs Boson Machine Learning Challenge that she helped organize in 2014, which officially required no knowledge of particle physics, had participants sift data from hundreds of thousands of simulated collisions (a small dataset by LHC standards) to infer which collisions contained a Higgs boson, the final subatomic particle from the Standard Model of particle physics for which evidence was observed.

The Higgs Boson Machine Learning Challenge, despite a modest top prize of \$7,000, attracted over 1,000 contenders, and in the end physicists were able to learn a thing or two from the data scientists, such as the use of cross-validation to avoid the problem of overfitting a model to just one or two datasets, according to Guyon. But even before this high-profile competition, high-energy physicists working at LHC had been using machine-learning tools to hasten their research. The

finding of the Higgs boson is a case in point. “The Higgs boson discovery employed a lot of the machine learning techniques,” says Maria Spiropulu, a physics professor at the California Institute of Technology (Caltech) who co-leads a team on the Compact Muon Solenoid (CMS), one of the main experiments at the LHC. “In 2005, we were expecting we would have a discovery with 14 TeV [tera electron Volts] around 2015 or 2016, and the discovery happened in 2012, with half the energy and only two years of data. Of course, we were assisted with nature because the Higgs was there, but in large part the discovery happened so fast because the computation was a very, very powerful ally.”

These days, machine learning techniques—mainly supervised learning—are used at every stage of the collider's operations, says Mauro D'Onofrio, a physicist at ETH Zurich who has worked on both the ATLAS particle physics experiment and CMS, the two main experiments at the LHC. That

process starts with the trigger system, which immediately after each collision determines if information about the event is worth keeping (most is discarded), and goes out to the detector level, where machine learning helps reconstruct events. Farther down the line, machine learning aids in optimal data placement by predicting which of the datasets will become hot; replicating these datasets across sites ensures researchers across the Worldwide LHC Computing Grid (a global collaboration of more than 170 computer centers in 42 countries) have continuous access to even the most popular data. “There are billions of examples [of machine learning] that are in this game—it is everywhere in what we do,” Donegà says.

At the heart of the collider’s efforts, of course, is the search for new particles. In data processing terms, that search is a classification problem, in which machine-learning techniques such as neural networks and boosted decision trees help physicists tease out scarce and subtle signals suggesting new particles from the vast “background,” or the multitude of known, and therefore uninteresting, particles coming out of the collisions. That is a difficult classification problem, says University of California, Irvine computer science professor Pierre Baldi, an ACM Fellow who has applied machine learning to problems in physics, biology, and chemistry.

“Because the signal is very faint, you have a very large amount of data, and the Higgs boson [for example] is a very rare event, you’re really looking for needles in a haystack,” Baldi explains, using most researchers’ go-to metaphor for the search for rare particles. He contrasts this classification problem with the much more prosaic task of having a computer distinguish male faces from female faces in a pile of images; that is obviously a classification problem, too, and classifying images by gender can, in borderline cases, be tricky, “but most of the time it’s relatively easy.”

These days, the LHC has no shortage of tools to meet the challenge. For example, one algorithm Donegà and his colleagues use focuses just on background reduction, a process whose goal is to squeeze down the background as much as possible so the signal can

## Spiropulu believes machine learning will enable physicists to push the frontier of their field beyond the Standard Model of particle physics.

stand out more clearly. Advances in computing—such as ever-faster graphics processing units and field-programmable gate arrays—have also bolstered the physicists’ efforts; these advances have enabled the revival of the use of neural networks, the architecture behind a powerful set of computationally intensive machine-learning techniques known as deep learning.

What is more, physicists have begun recruiting professional data scientists, in addition to continuing to master statistics and machine learning themselves, says Maurizio Pierini, a physicist at CERN who organized last November’s workshop on data science at the LHC. He points to ATLAS, the largest experiment at the LHC, which in the last two or three years has done “an incredible job of attracting computer scientists working with physicists,” he says.

Although machine learning has become an indispensable tool at the Large Hadron Collider, there is a natural tension in the use of machine learning in physics experiments. “Physicists are obsessed with the idea of understanding each single detail of the problem they’re dealing with,” says Pierini. In physics, “you take a problem and you decompose it to its ingredients, and you model it,” he explains. “Machine learning has a completely different perspective—you’re demanding your algorithm to explore your dataset and find the patterns between the different features in your dataset. A physicist would say, ‘No, I want to do this myself.’”

The opacity of what exactly goes on inside machine learning algorithms—

the “black-box problem,” as it is usually called—makes physicists uneasy, but they have found ways to increase their confidence in the results of this new way of doing science. “When it comes to very elaborate software that we don’t understand, we’re all skeptics,” says Spiropulu, the Caltech physicist, “but once you do validation and verification, and you’re confronted with the fact that these are actually things that work, then there’s nothing to say, and you use them.” Simply put, the physicists know the algorithms work because they have been successfully tested many times on known physics phenomena, so there is every reason to believe they work in general.

In fact, Spiropulu believes machine learning will enable physicists to push the frontier of their field beyond the Standard Model of particle physics, accelerating investigations of more mysterious phenomena like dark matter, supersymmetry, and any other new particles that might emerge in the collider.

“It’s not unleashing some magic box that will give us what we want. These algorithms are extremely well-architected; if they fail, we will see where they fail, and we will fix the algorithm.” ■

### Further Reading

Adam-Bourdarios, C., Cowan, G., Germain, C., Guyon, I., Kégl, B., and Rousseau, D. The Higgs boson machine learning challenge. *JMLR: Workshop and Conference Proceedings* 2015 <http://jmlr.org/proceedings/papers/v42/cowa14.pdf>

Baldi, P., Sadowski, P., and Whiteson, D. Searching for exotic particles in high-energy physics with deep learning, *Nature Communications* (2014), Vol 5, pp1–9 <http://bit.ly/1RCaiOB>

Donegà, M. ML at ATLAS and CMS: setting the stage. Data Science @ LHC 2015 Workshop (2015). <https://cds.cern.ch/record/2066954>

Melis, G. Challenge winner talk. Higgs Machine Learning Challenge visits CERN (2015). <https://cds.cern.ch/record/2017360>

CERN. The Worldwide LHC Computing Grid. <http://home.cern/about/computing/worldwide-lhc-computing-grid>

Based in San Francisco, **Marina Krakovsky** is the author of *The Middleman Economy: How Brokers, Agents, Dealers, and Everyday Matchmakers Create Value and Profit* (Palgrave Macmillan, 2015).

# Booming Enrollments

*The Computing Research Association works to quantify the extent, and causes, of a jump in undergraduate computer science enrollments.*

**A**T THE COMPUTING Research Association (CRA) Snowbird conference in 2014, Jim Kurose (then at University of Massachusetts-Amherst) and Ed Lazowska (University of Washington) presented a session on burgeoning enrollments in U.S. computing courses. In response, CRA's Board formed a committee to further study enrollment-related issues, chaired by CRA board member Tracy Camp.

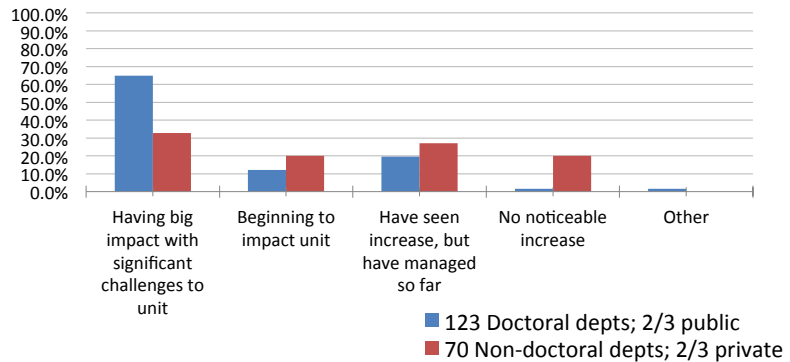
A panel on the upsurge in undergraduate computer science (CS) enrollments in the U.S. took place at the ACM Special Interest Group on Computer Science Education Technical Symposium last year (SIGCSE 2015); shortly thereafter, the full committee went to work with the goal of measuring, assessing, and better understanding enrollment trends and their impact, with a special focus on diversity.

Explained Susan B. Davidson, CRA Board Chair and a member of the CRA enrollments committee, "Over the past few years, computing departments across the country have faced huge increases in course enrollments. To understand the extent and nature of these 'booming enrollments,' CRA has undertaken a study that surveys both CRA-member doctoral departments as well as ACM non-doctoral departments."

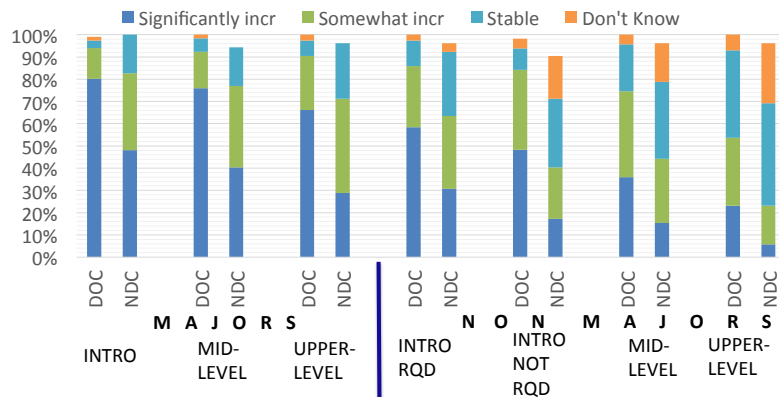
In addition to attempting to identify the extent of the "boom" in CS enrollments, Davidson said, "We are trying to understand which students are making up this boom: CS majors? Students from other fields seeking to minor in CS? Students in other fields taking a course or two in CS? And why are they doing so; what is driving them?"

The study also aims to determine how academic departments are coping with such a boom, Davidson said. "Are they restricting enrollments and, if so, what is the impact on the diversity of students enrolled? Are they increasing

## TO WHAT EXTENT ARE INCREASING UG ENROLLMENTS IMPACTING YOUR UNIT?



## IN WHICH COURSES IS DEMAND INCREASING?



From Booming Enrollments Survey Data preliminary results presentation, Computing Research Association.

class sizes and, if so, what is the impact on quality of instruction? Are they increasing faculty sizes? What other strategies are being used?"

The hope, Davidson said, "is that answers to these questions will give university administrators and computing departments insights into the extent of the boom, and enable them to develop better strategies to managing booming enrollments."

The study includes data acquired from several sources, including sources involved in the annual CRA Taulbee survey (the principal source of infor-

mation on the enrollment, production, and employment of Ph.D.'s in computer science and computer engineering (CE), also providing salary and demographic data for CS and CE faculty in North America), and sources from the annual ACM NDC Study of Non-Doctoral Granting Departments in Computing. In addition to surveying institutions, data was collected from students via a Fall 2015 student survey by the CRA Center for Evaluating the Research Pipeline (CERP).

The slide deck for the preliminary results of these surveys may be seen at

[http://inside.mines.edu/~tcamp/SIGC-SE2016\\_Boom.pdf](http://inside.mines.edu/~tcamp/SIGC-SE2016_Boom.pdf)

Among the preliminary results to be gleaned from the institutions surveyed (key findings will be presented at CRA Snowbird 2016, with a final report planned for the fall):

- About two-thirds of 123 doctoral departments and one-third of 70 non-doctoral departments surveyed reported increasing undergraduate enrollments were having a big impact on them, resulting in significant challenges.

- About 80% of doctoral programs reported significant increases in demand for introductory courses in a CS or CE major; less than half of participating non-doctoral programs reported similarly significant increases.

- Increases in undergraduate enrollments were seen as creating problems in at least 40% of the departments surveyed. Most (78%) doctoral departments had issues with classroom space, followed by the availability of sufficient faculty (69%), sufficient teaching assistants (61%), and faculty workloads (61%). In non-doctoral programs, the most frequently reported concerns were sufficient faculty (44%) and faculty workload (42%).

- In response to those concerns, more than 80% of doctoral departments increased the size of classes and the number of sections offered during the academic year. More than 40% of non-doctoral departments reported increasing class size, and more than 60% reported increasing the number of sections offered in a school year.

- In terms of staffing, more than 70% of doctoral departments reported increased use of undergraduate teaching assistants, while more than 60% reported the increased use of adjuncts/visiting faculty, having graduated students teach, or increasing the teaching faculty. More than 40% of non-doctoral departments reported increased use of adjuncts/visiting faculty; another 42% said they would like to expand their tenure-track faculty, but cannot.

- In the context of diversity, no adverse effects on recruitment or retention were reported, but only 35%–40% of responding departments said they explicitly consider the impact on diversity when choosing actions. Diversity concerns have not prevented or nul-

## **“The survey responses are giving us a lot of information on how universities are handling the ‘boom’ and what the biggest concerns are.”**

lified any enrollment-related actions taken in those departments.

The student survey received responses from 2,477 students, 98% of whom had enrolled in an introductory CS class; 72% of them were computing majors, 7% were computing minors, and the balance were either undeclared or had declared non-computing majors or minors. The gender mix was roughly 2:1 men to women.

The survey found most (86%) enrolled in an introductory computing class because “it was required for my major/minor.” The next most frequent response, “curiosity or interest in computers,” was reported by 39% of respondents.

When the 55 respondents who had dropped an introductory computing course were asked why, nearly half

(46%) said it was too challenging; 75% of those were women. About 29% (20% of men and 45% of women) said they dropped the class because they did not enjoy the professor’s teaching, and 26% said they were “no longer interested in computers.”

Tracy Camp, professor of computer science in the Department of Electrical Engineering and Computer Science at the Colorado School of Mines, and past co-chair of CRA’s Committee on the Status of Women in Computing Research (CRA-W), observed, “The survey responses are giving us a lot of information on how universities are handling the ‘boom’ and what the biggest concerns are. We are also learning why students are so interested in taking an intro to computing course.”

ACM president Alexander L. Wolf said the study’s findings were critical to ensuring North American universities are prepared to handle the growing numbers of students who will enter computer-related degree programs in the coming years. “Particularly in the context of President Obama’s Computer Science for All initiative, we are going to see enrollments in computer-related classes continue to skyrocket.”

Wolf added, “But this is not a phenomenon specific to the U.S.; rather, we’re seeing booming interest in CS education around the world.” **C**

**Lawrence M. Fisher** is Senior Editor/News for ACM magazines.

© 2016 ACM 0001-0782/16/07 \$15.00

### **Why are students dropping introductory computer science classes?**



# Legal Advice on the Smartphone

*New apps help individuals contest traffic, parking tickets.*

**S**UCCESSFULLY CHALLENGING a summons or ticket can be challenging and time-consuming. Laypeople must understand the violation that has occurred, determine the type of defense that is legally acceptable, and learn the type of documentation most often used successfully to win a dismissal or reduction of that violation in that specific jurisdiction. For the average person, the time and effort to acquire such arcane knowledge often outweighs the desire to fight the ticket.

Not surprisingly, applications designed to help access and navigate through the legal system are quickly gaining favor among the public. From apps that help you fight parking and traffic tickets to apps that report violations, mobile technology apps are stepping in to serve as an on-the-go legal assistant. The use of a combination of algorithms, technology, and specialized industry experience offer a more-efficient experience dealing with, and within, the legal system.

Among drivers' greatest annoyances are parking and traffic tickets. The process of fighting a ticket traditionally has included conducting the research to determine whether or not a ticket might be dismissible; identifying, collecting, and submitting the proper evidence, and then going through a time-consuming process to actually appeal the ticket, either via mail or by heading to traffic court to fight the ticket in person.

Enter WinIt (appwin.it.com), a mobile app available on the iOS and Android platforms that helps users by providing an algorithm that identifies the type of ticket, and then automatically provides a list of the evidence or documentation that is most likely to help convince a judge to dismiss a ticket or reduce a fine. After the user provides the documentation, WinIt taps



into the parking-law experts of Empire Commercial Services, a company that has fought commercial parking violations for its customers in New York City for more than 25 years. These specialists will then review the ticket (as parking tickets in New York City have both a pre-printed section and a section that is filled out by hand by the parking officer) and the supporting documentation, and will contest the ticket on the user's behalf in court.

Once the app has been downloaded, users simply need to take a clear picture of the ticket, and then submit it to WinIt. The benefit to users is being able to quickly determine whether or not it will be worth the time to fight a ticket, via the app's algorithm, which prompts users to scan and attach the documentation that would be required to successfully fight a violation (such as a parking receipt, a photo of the vehicle's position, or a copy of the vehicle registration). If a user cannot provide such documentation, WinIt says the likelihood a ticket will be dismissed is very low.

"The city has always given individuals the ability to fight their tickets, whether through the mail, in person, or online," says Christian Fama, a co-owner of WinIT and an executive at Empire Commercial Services. "If a ticket isn't dismissible, the judge doesn't dismiss it." However, the app, which allows one to see what types of defenses are available, provides users a way to easily evaluate whether a challenge to a ticket is likely to be successful without going through the entire process of contesting a ticket.

If WinIT gets the ticket dismissed, the user pays the company 50% of the value of the fine; if the ticket is not dismissed, the user simply needs to pay the fine in full, and owes WinIT no fee.

WinIT began its testing phase in March 2015, and became available for public use three months later. WinIT currently processes hundreds of tickets a day, and credits the success of the app to Empire's decades of success with fighting tickets for commercial clients.

Fama says WinIT's success rate has exceeded his initial expectations, and

even exceeded the typical 20%–25% win rate of Empire's commercial ticket-fighting business.

While the service is only available in New York City at present, the company hopes to expand to other jurisdictions in the future, and to get into the business of contesting moving violations.

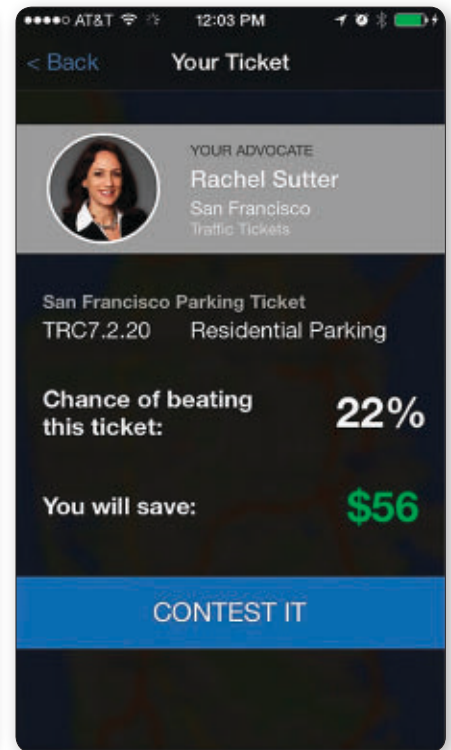
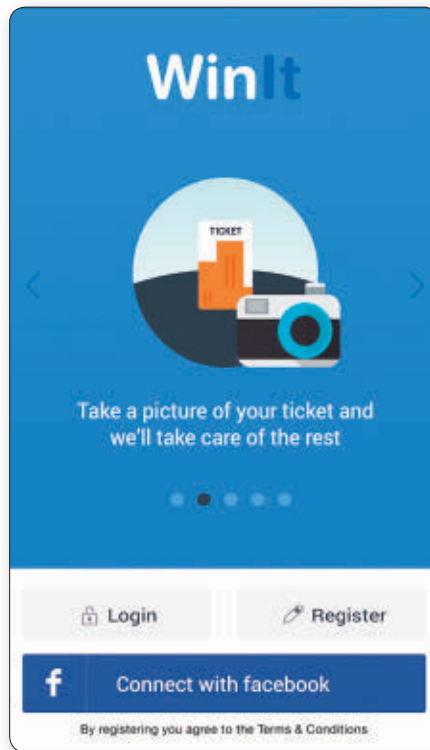
Another provider based in California has already made that move into moving violations, driven in part by push-back from the cities of Los Angeles, Oakland, and San Francisco. Launched in 2012, Fixed was founded by David Hegarty to streamline the process of fighting parking tickets by capturing ticket data, submitting the challenge to the respective municipal parking authority, and matching app users with attorneys, if so desired. However, in mid-2015, the aforementioned cities made a software change that blocked bulk electronic submissions of parking tickets, making it very difficult for Fixed to effectively submit and contest tickets through its app, since they would have to be sent one by one.

As a result, "we've suspended parking ticket operations in all areas," Hegarty says, noting the company is now solely focused on expanding its traffic ticket business. "We're currently operating in over a dozen states, and hope to get to 25-plus [states] in the next year."

Like the parking ticket service, the traffic ticket service matches tickets with attorneys using a learning algorithm that is fed new data from each ticket submitted, such as the location of the violation, the type of violation, and relevant data such as the vehicle's speed or other situational data related to the alleged traffic violation. This algorithm then matches the circumstances of the ticket with attorneys that have specific experience with the type of violation in that specific geographic area.

Still, traffic tickets and parking tickets are just scratching the surface of the plethora of legal issues that often come up. Los Angeles-based LegalTap (legaltap.co) was founded to make accessing and consulting with lawyers simpler, more convenient, and less expensive than simply flipping through the Yellow Pages or doing a random search online.

The application is designed to provide efficient, cost-effective legal consultation and forms for routine legal



Screenshots from WinIT (left) and Fixed, two apps that help users contest parking tickets.

issues, including business start-up issues, immigration issues, name changes, or other relatively straightforward topics that generally still require some legal input or review.

Natacha Gaymer-Jones, the founder of LegalTap, says the app facilitates a 15-minute consultation with a lawyer for just \$39, with the option to schedule a more in-depth conversation at a later date. The application was launched on iOS and Android in June 2015, and as of February 2016, has generated more than 1,500 calls on topics ranging from parking ticket issues and immigration issues to business issues and more.

"We wanted to provide people with a way to address quick legal issues, by providing an app that's accessible and also at a price point that was understandable," Gaymer-Jones says, noting the application also helps lawyers quickly vet potential clients to see if they are a good fit, and provides "an easy way for lawyers to make money without giving away a free hour of legal consultation."

The app features a pre-programmed algorithm that matches the type of query with lawyers that have signed up to be part of the LegalTap network. The system is designed to match only attorneys qualified in the appropriate area of law to clients' needs; a question on

divorce law will only be routed to vetted divorce attorneys.

Once a call comes in and is analyzed, lawyers that have agreed to answer calls have 90 seconds to take the call, or the call is routed to another qualified attorney. If a call is accepted, the lawyer can either handle it then (and be paid around \$150 per hour for handling incoming calls), or may choose to schedule an in-person meeting with the caller, thereby gaining a new client.

LegalTap also offers a form shop where users can select and download basic legal forms, fill them out, and then consult with a LegalTap attorney before submitting them. "A lot of law firms are trying to productize their offerings," Gaymer-Jones says. "We want to give a bit of personal advice."

Still, mobile apps are not simply tools for fighting tickets or managing legal problems. An app released in 2009 by Parking Mobility (www.parkingmobility.com), a non-profit group dedicated to addressing accessible parking abuse, lets users capture and report instances of people who illegally park in handicapped-accessible parking spots.

The project director, Mack Marsh, who was paralyzed in an accident 15 years ago, founded the group because he and others in the disability community

felt accessible parking violations were not adequately being addressed by law enforcement. However, the app's appeal extends beyond those who are directly or indirectly affected by disabilities.

"We have people who don't identify with the disability community—they just see this as an issue," Marsh says. "Typically, they download the app because they are angry when they see a violation, and they see there's no other way" to adequately address the issue, since these parking violations are often of low priority to police departments.

Once downloaded, the app asks users to capture three photos of an alleged violation (one from the front capturing the lack of an accessible-parking placard, one from the rear to capture the license plate, and a third photo capturing the violation itself).

Marsh says the app makes it easier to collect verified instances of accessible parking violations, since it captures photos and stamps them with the phone's geolocation data, along with time and date information.

Parking Mobility currently has more than 500,000 users worldwide, broken into two groups. Casual users can be located anywhere, and their violation reports are automatically collected by Parking Mobility, which then passes along the data to the relevant municipalities to highlight the prob-

## The Parking Mobility App documents accessible parking violations by stamping photos of the violations with the phone's geolocation data, along with date and time.

lem of accessible parking violations, with the eventual goal of creating a partnership between the jurisdiction and a second group of users, known as citizen volunteers, who are deputized and allowed to issue parking citations via the app.

"Every report that comes into our system is reviewed by our staff or one of our board members," Marsh says, noting any reports submitted by casual users that do not capture enough or proper evidence of a violation are rejected, along with an explanation

of why the submission was rejected. However, for users that have been gone through the training and have been deputized by their local law enforcement department, submissions must "have to have the elements that can stand up in court."

Currently, Parking Mobility has partnerships with jurisdictions in Texas, Kansas, and Oregon, and is negotiations with jurisdictions in Florida, Georgia, North and South Carolina, Arizona, and Colorado.

Still, whether a partnership has been struck or not, "we encourage the use of the Parking Mobility App everywhere," Marsh says. "Reports in no-partner communities are as important, if not more so, than reports which result in citations, [because] data from reports is the only way to get communities to issue citations." 

### Further Reading

**Fixed Blocked in Three Cities:** <https://nakedsecurity.sophos.com/2015/10/15/fixed-app-that-fights-parking-tickets-blocked-in-3-cities/>

**Parking Mobility:** <https://www.youtube.com/watch?v=Cax5yVyC8>

Keith Kirkpatrick is principal of 4K Research & Consulting, LLC, based in Lynbrook, NY.

© 2016 ACM 0001-0782/16/07 \$15.00

### Milestones

## Computer Science Awards, Appointments

### 2016 NAE MEMBERS INCLUDE COMPUTER SCIENTISTS

The National Academy of Engineering (NAE) recently elected 80 new members and 22 new foreign members.

NAE membership honors those who have made outstanding contributions to "engineering research, practice, or education, including, where appropriate, significant contributions to the engineering literature," and to "the pioneering of new and developing fields of technology, making major advancements in traditional fields of engineering, or developing/implementing innovative approaches to engineering education."

The following computer scientists were among the 80

newest NAE members:

- ▶ Thomas E. Anderson, Warren Francis and Wilma Kolm Bradley Endowed Chair in Computer Science and Engineering, University of Washington.
- ▶ Dan Boneh, professor of computer science and electrical engineering, Stanford University.
- ▶ Frederick R. Chang, director of the Darwin Deason Institute for Cyber Security, Bobby B. Lyle Endowed Centennial Distinguished Chair in Cyber Security, and professor in the department of computer science and engineering, Lyle School of Engineering, Southern Methodist University.
- ▶ Albert G. Greenberg, distinguished engineer and

director, Azure Networking, Microsoft Corp.

- ▶ Mehdi Hatamian, senior vice president of engineering, Broadcom Corp.
- ▶ Mary Cynthia Hipwell, vice president of engineering, Bühler, Plymouth, MN.
- ▶ Paul E. Jacobs, executive chairman, Qualcomm Inc.
- ▶ Anil K. Jain, University Distinguished Professor, department of computer science and engineering, Michigan State University.
- ▶ David S. Johnson, visiting professor of computer science, Columbia University.
- ▶ Charles E. Leiserson, Edwin Sibley Webster Professor, department of electrical engineering and computer

science, Massachusetts Institute of Technology.

- ▶ Bruce G. Lindsay, IBM Fellow Emeritus, IBM Almaden Research Center.
  - ▶ Arati Prabhakar, director, U.S. Defense Advanced Research Projects Agency.
  - ▶ John R. Treichler, president, Raytheon Applied Signal Technology.
  - ▶ Stephen M. Trimberger, fellow, Xilinx, Inc..
- One new NAE foreign member also is focused on computer science:
- ▶ Geoffrey E. Hinton, distinguished emeritus professor, department of computer science, University of Toronto, and distinguished researcher, Google Inc.



DOI:10.1145/2935878

Pamela Samuelson

## Legally Speaking

# *Apple v. Samsung* and the Upcoming Design Patent Wars?

*Assessing an important recent design patent infringement court decision.*

UNTIL RECENTLY, DESIGN patents have been a relatively obscure category of U.S. intellectual property (IP) rights. Design patent law was originally intended to encourage investments in novel and inventive ornamental designs for articles of manufacture, such as carpets and lamps. However, many design patents have been granted in recent years to makers of advanced information technologies, such as smartphones. Often these are for component parts of their technologies, such as product configurations and virtual designs embedded in software.

This column reviews the *Apple v. Samsung* design patent infringement case. It considers the two key issues Samsung brought to the Supreme Court for review. One set concerns the scope of design patents and the test that should be used to judge infringement. A second concerns whether the infringer of a design patent must disgorge all of its profits from the sale of

a product in which the design is embodied, or only those profits that are attributable to the infringement. The Supreme Court has decided to address the second issue, but not the first.

### **Apple v. Samsung**

The most recent of several litigations between Apple and Samsung involves three design patents that cover specific portions of the external configuration of smartphone designs: a black rectangular round-cornered front face for the device; a substantially similar rectangular round-cornered front face with a surrounding rim or bezel; and a colorful grid of 16 icons to be displayed on a screen.

Apple sued Samsung for infringing these patents, as well as for infringing trade dress rights in the external design of the iPhone. A jury found that Samsung had infringed both Apple's design patents and trade dress rights and ordered Samsung to pay \$930 million in damages for these infringements.

The Court of Appeals for the Federal Circuit (CAFC) overturned the trade dress claim saying that the external design of the Apple smartphone was too functional to qualify for trade dress protection. The rounded corners and bezel were, it ruled, designed to make smartphones more "pocketable" and to protect against breakage when the phone was dropped. The icon displays promote usability by communicating to the user which functionalities they can invoke by touching the icons.

The Supreme Court has held that trade dress, such as product configurations, is too functional to be protectable "if it is essential to the use or purpose of the article or if it affects the cost or quality of the article, or would put competitors at a significant non-reputational-related disadvantage." Under this standard, the Apple trade dress claims failed.

However, the CAFC affirmed the design patent infringement ruling, rejecting Samsung's argument that the same features the CAFC thought were too functional for trade dress protection made them ineligible for design patenting too. The CAFC disagrees with the proposition that the functionality test for design patents is the same as for trade dress.

Somewhat to Samsung's relief, the CAFC ordered the damage award to be cut to \$399 million. But even this amount, Samsung argues, is excessive because it represents all of the profits that Samsung has made in selling the phones embodying the patented designs.

Samsung asked the Supreme Court to review the design patent ruling and the monetary damage award. It raised two questions for the Court's consideration:

- Where a design patent includes unprotected non-ornamental features, should a district court be required to limit that patent to its protected ornamental scope?

- Where a design patent is applied to only a component of a product, should an award of infringer's profits be limited to those profits attributable to the component?

### Ornamentality and Filtration

Samsung challenged the Apple design patent infringement claims because some aspects of the patented designs are too conceptual or functional to be within the valid scope of the patents. Geometric shapes, for instance, are concepts in the public domain.

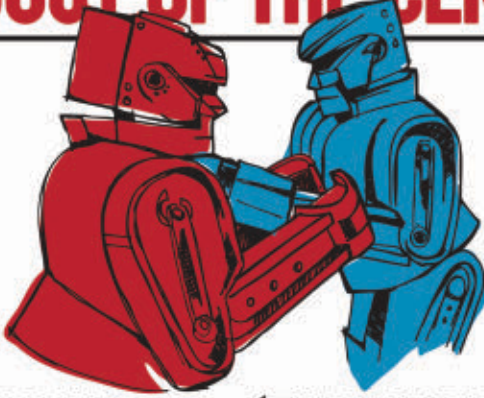
Samsung's brief emphasized that virtually the same design features that the CAFC held were too functional to be eligible for trade dress protection were also the basis for the design patent claims. The finding of functionality that vitiated Apple's trade dress claim should, Samsung argued, have defeated the design patent claim as well. Functional designs should be protected, if at all, by utility patents, not by design patents.

The CAFC, however, regards a design to be too functional to qualify for a patent only if that design is solely dictated by functionality, such that no other design options exist. Samsung argued that is too stringent a test for functionality on the design patent side. After all, a design is supposed to be ornamental to qualify for design patent protection. Ornamental designs, almost by definition, will not be dictated by function.

Samsung's second challenge to the infringement ruling concerned the overbroad test for infringement the lower court had used and the CAFC endorsed. That test focused on whether an ordinary observer would find similarities between the designs to be so substantial that they would induce customers to buy the alleged infringer's product instead of the patentee's product. As applied in the *Apple* case, Samsung argued that this test failed to filter out of consideration elements of smartphone design not covered by Apple's design patents.

# IP BATTLE

## THE BOUT OF THE CENTURY



DESIGNER

**APPLE**

VS

'THE INFRINGER'

**SAMSUNG**

**ARGUMENTS**

WILL BE PRESENTED!

**JUDGEMENTS**

WILL BE DECIDED!

**DAMAGES**

TO BE AWARDED!

## THE TIME HAS FINALLY ARRIVED!

★ SUPREME COURTSIDE IS WHERE IT'S AT ★

Samsung complained this test failed to filter out the conceptual and functional elements of the patented designs. The court also directed the jury to make a judgment based on their overall impressions of the appearance of the Apple and Samsung smartphones. Under this approach, the jury could have decided that the overall similarities between the Apple and Samsung smartphones justified the infringement finding, even though the design patents only covered a small number of features.

A coalition of high-technology companies, including Dell, eBay, Facebook, and Google, filed a brief in support of Samsung's petition for Supreme Court review. That amicus curiae (friend of the court) brief criticized the CAFC for failing to grasp the complexities of today's highly componentized technologies.

That brief pointed out that today's information technologies are much

more complex and have many more component parts than the manufactured products Congress envisioned as suitable for design patent protection back in 1842 when the design patent law was first enacted.

It is one thing to ask a jury to consider whether their overall impression of a design of an article of manufacture, such as a competing carpet, infringed a design patent. The patented design would presumably cover the feature that created most of the value in the patentee's carpet.

However, high-tech products such as smartphones have a staggering number of functional and design features and component parts. As of 2012, more than 250,000 patents had been issued for smartphone-related inventions. Samsung argued that a test for design patent infringement should focus on the similarities in the overall impression of the feature covered by the design patent, not the product

as a whole. Although Samsung's arguments have some merit, the Supreme Court has decided not to review either the functionality issue or the proper test for infringement in the *Apple v. Samsung* case.

### Total Profits As Windfall

An even more urgent concern driving Samsung's plea for Supreme Court review arises from the CAFC's approval of an award of all of its profits from selling the smartphones that infringed those three design patents.

The CAFC acknowledged that a total profits award for infringement of the design patents in this case was difficult to justify as a matter of equity. However, it held that the statute required approval of a total profits award.

It relied on this part of the relevant statute, 35 U.S.C. § 289: "Whoever during the term of a patent for a design, without license of the owner ... applies the patented design ... to any article of manufacture ... shall be liable to the owner to the extent of his total profit, but not less than \$250."

The statute plainly speaks about "total profit" as a suitable award for infringement of a design patent. But what is the relevant "article of manufacture?"

The CAFC decided the relevant article of manufacture in *Apple v. Samsung* was the smartphone itself, not just the subparts covered by the design patents. After all, no one would buy only the design-patented screen with icons or round-shaped rim with a bezel. People buy a whole smartphone. This explains why the CAFC thought the smartphone was the article of manufacture whose total profits courts must award when design patents are infringed.

This interpretation of design patent awards is inconsistent with principles that guide damage awards in other types of IP cases. Had Samsung infringed a utility patent, a copyright, or protectable trade dress, an award of monetary damages would be based on the harm that was attributable to the infringement. A total profits award for utility patent infringement, for instance, would only be available if there was evidence the patented feature was responsible for the market demand for the product embodying it.

## This interpretation of design patent awards is inconsistent with principles that guide damage awards in other types of IP cases.

The general rule is that IP owners are entitled to compensation for harm caused by infringement. They are not entitled to windfall awards when some or virtually all of the value of a product is due to aspects not covered by an IP right. The CAFC's ruling is inconsistent with conventional rules of IP law and with sound principles of equity.

Samsung's and amicus briefs filed in support of its petition for Supreme Court review have pointed to earlier appellate court decisions that apportioned damages for infringement of design patents.

One case involved a design patent on a piano case. The court approved an award of the infringer's profits from use of the patented design in a piano case, but disapproved the argument that the infringer should have to disgorge its profits from sale of pianos containing the patented design. "[R]ecover should be confined to the subject of the patent."

The CAFC's interpretation of § 289 is also arguably inconsistent with another part the CAFC did not seem to heed. It states that the patentee "shall not twice recover the profit *made from the infringement*." This is a limiting principle that links the award of damages to the infringement.

There is also a question of fairness. How could it be fair, Samsung asked, for a court or jury to award 100% of a firm's profits for infringement of one design patent if the patented feature accounted for only 1% of the value of the product? And what if a second design patent owner came along and a jury found infringement of that pat-

ent too? Would a second award of total profits be fair, or would the first patentee's windfall have exhausted the available damages?

More concretely, consider this hypothetical. Apple owns a design patent on the musical note icon for smartphones. Samsung is not charged with infringing that patent. But suppose the only design claim against Samsung pertained to that patent.

An IP professors amicus brief in support of Samsung's petition (written by Stanford's Mark Lemley and joined by yours truly) pointed out that it would not be reasonable to award the same \$399 million in total profits for infringement of that one patent.

Demand for iPhones is driven by many factors. But the music icon is a very small part of the value of any smartphone that might embody it. Proportionality should apply in all awards for monetary compensation for infringing IP rights.

Several amicus briefs filed in support of Samsung's petition for Supreme Court review warned that if the Court failed to overturn the "total profits" award ruling in this case, this would set off a new round of patent troll litigations. This would be harmful to innovation and competition in high-tech industries, especially given the low quality of some issued design patents.

### Conclusion

The Supreme Court has decided to review the total profits recovery question raised by Samsung. On that point, Samsung seems likely to prevail. Differentiating between "ornamental" and "functional" elements of designs for articles of manufacture is a trickier matter, but surely the test for infringement of a design patent should focus on that design rather than products as a whole. Unfortunately, the Court decided not to review this important question. The *Apple v. Samsung* case is a very important one. Resolved well, it will mitigate design patent wars in high-tech industries. Resolved badly, it will surely spark such wars. 

**Pamela Samuelson** (pam@law.berkeley.edu) is the Richard M. Sherman Distinguished Professor of Law and Information at the University of California, Berkeley.

Copyright held by author.

# Historical Reflections

## How Charles Bachman Invented the DBMS, a Foundation of Our Digital World

*His 1963 Integrated Data Store set the template for all subsequent database management systems.*

**F**IFTY-THREE YEARS AGO a small team working to automate the business processes of the General Electric Company built the first database management system. The Integrated Data Store—IDS—was designed by Charles W. Bachman, who won the ACM's 1973 A.M. Turing Award for the accomplishment. Before General Electric, he had spent 10 years working in engineering, finance, production, and data processing for the Dow Chemical Company. He was the first ACM A.M. Turing Award winner without a Ph.D., the first with a background in engineering rather than science, and the first to spend his entire career in industry rather than academia.

Some stories, such as the work of Babbage and Lovelace, the creation of the first electronic computers, and the emergence of the personal computer industry have been told to the public again and again. They appear in popular books, such as Walter Isaacson's recent *The Innovators: How a Group of Hackers, Geniuses and Geeks Created the Digital Revolution*, and in museum exhibits on computing and innovation. In contrast, perhaps because database management systems are rarely experienced directly by the public,

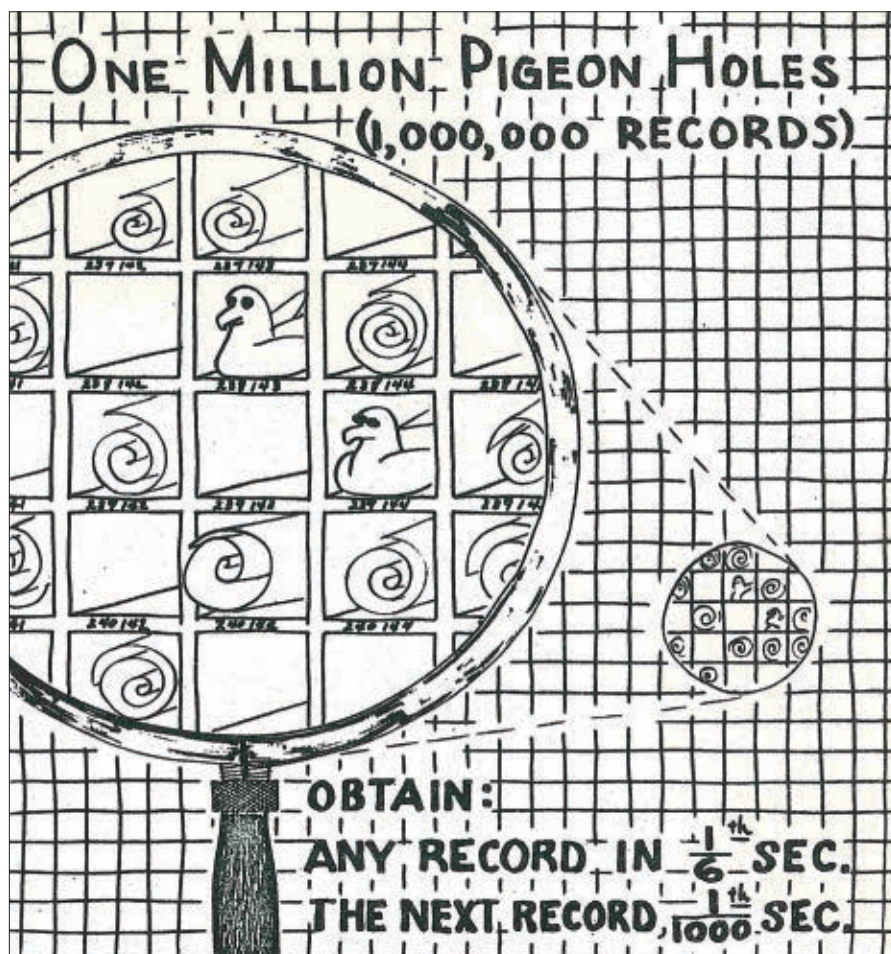


Figure 1. This image, from a 1962 internal General Electric document, conveyed the idea of random access storage using a set of “pigeon holes” in which data could be placed.

database history has been largely neglected. For example, the index of Isaacson's book does not include entries for "database" or for any of the four people to have won Turing Awards in this area: Charles W. Bachman and Edgar F. Codd (1981), James Gray (1988), or Michael Stonebraker (2014).

That's a shame, because if any technology was essential to the rebuilding of our daily lives around digital infrastructures, which I assume is what Isaacson means by "the Digital Revolution," then it was the database management system. Databases undergird the modern world of online information systems and corporate intranet applications. Few skills are more essential for application developers than a basic familiarity with SQL, the standard database query language, and a database course is required for most computer science and information systems degree programs. Within ACM, SIGMOD—the Special Interest Group for Management of Data—has a long and active history fostering database research. Many IT professionals center their entire careers on database technology: the census bureau estimates the U.S. alone employed 120,000 database administrators in 2014 and predicts faster than average growth for this role.

Bachman's IDS was years ahead of its time, implementing capabilities that had until then been talked about but never accomplished. Detailed functional specifications for the system were complete by January 1962, and Bachman was presenting details of the planned system to his team's in-house customers by May of that year. It is less clear from archival materials when the system first ran, but Bachman tells me that a prototype installation of IDS was tested with real data in the summer of 1963, running twice as fast as a custom-built manufacturing control system performing the same tasks.

The details of IDS, Bachman's life story, and the context in which it arose have been explored elsewhere.<sup>2,6</sup> In this column, I focus on two specific questions:

- Why do we view IDS as the first database management system, and
- What were its similarities and differences versus later systems?

There will always be an element

## If any technology was essential to the rebuilding of our daily lives around digital infrastructures, it was the database management system.

of subjectivity in judgments about "firsts," particularly as IDS predated the concept of a database management system. As a fusty historian I value nuance and am skeptical of the idea that any important innovation can be fully understood by focusing on a single breakthrough moment. I have documented many ways in which IDS built on earlier file management and report generation systems.<sup>7</sup> However, if any system deserves the title of "first database management system" then it is clearly IDS. It became a model for the earliest definitions of "data base management system" and included most of the core capabilities later associated with the concept.

### What Was IDS For?

Bachman created IDS as a practical tool, not an academic research project. In 1963 there was no database research community. Computer science was just beginning to emerge as an academic field, but its early stars focused on programming language design, theory of computation, numerical analysis, and operating system design. In contrast to this academic neglect, the efficient and flexible handling of large collections of structured data was the central challenge for what we would now call corporate information systems departments, and was then called business data processing.

During the early 1960s the hype and reality of business computing diverged dramatically. Consultants, visionaries, business school professors, and computer salespeople had all agreed that the best way to achieve real economic payback from computerization was to

establish a "totally integrated management information system."<sup>8</sup> This would integrate and automate all the core operations of a business, ideally with advanced management reporting and simulation capabilities built right in. The latest and most expensive computers of the era had new capabilities that seemed to open the door to a more aggressive approach. Compared to the machines of the 1950s they had relatively large memories. They featured disk storage as well as tape drives, could process data more rapidly, and some were even used to drive interactive terminals.

The reality of data processing changed much more slowly than the hype, and remained focused on simple administrative applications that batch processed large files to accomplish tasks such as weekly payroll processing, customer statement generation, or accounts payable reporting.

Many companies announced their intention to build totally integrated management information systems, but few ever claimed significant success. A modern reader would not be shocked to learn that firms were unable to create systems of comparable scope to today's Enterprise Resources Planning and data warehouse projects using computers with perhaps the equivalent of 64KB of memory, no real operating system, and a few megabytes of disk storage. Still, even partially integrated systems covering significant portions of a business would have real value. The biggest roadblocks to even modest progress toward this goal were the sharing of data between applications and the difficulties application programmers faced in exploiting random access disk storage.

Getting a complex job done might involve dozens of small programs and the generation of many working tapes full of intermediate data. These banks of whirring tape drives provided computer centers with their main source of visual interest in the movies of the era. Tape-based processing techniques evolved directly from those used with pre-computer mechanical punched card machines: files, records, fields, keys, grouping, merging data from two files, and the hierarchical combination of master and detail records within a single file. These applied to

magnetic tape much as they had done to punched cards, except that tape storage made sorting much harder. The formats of tape files were usually fixed by the code of the application programs working with the data. Every time a field was added or changed all the programs working with the file would need to be rewritten. If applications were integrated, for example, by treating order records from the sales accounting system as input for the production scheduling application, the resulting web of dependencies made it increasingly difficult to make even minor changes when business needs shifted.

The other key challenge was making effective use of random access storage in business application programs. Sequential tape storage was conceptually simple, and the tape drives themselves provided some intelligence to aid programmers in reading or writing records. Applications were batch-oriented because searching a tape to find or update a particular record was too slow to be practical. Instead, master files were periodically updated with accumulated data or read through to produce reports. With the arrival, in the early 1960s, of disk storage a computer could theoretically apply updates one at a time as new data came in and generate reports as needed based on current data. Indeed this was the target application of IBM's RAMAC computer, the first to be equipped with a hard disk drive. A programmer working with a disk-based system could easily instruct the disk drive to pull data from any particular platter or track, but the hard part was figuring out where on the disk the desired record could be found. The phrase "data base" was associated with random access storage but was not particularly well established, so Bachman's alternative choice of "data store" would not have seemed any more or less familiar at the time.

Without significant disk file management support from the rudimentary operating systems of the era only elite programmers could hope to create an efficient random access application. Mainstream application programmers were beginning to shift from assembly language to high-level languages such as COBOL, which included high-level

support for structuring data in tape files but lacked comparable support for random access storage. Harnessing the power of disks meant finding ways to sequence, insert, delete, or search for records that did not simply replicate the sequential techniques used with tape. Solutions such as hashing, linked lists, chains, indexing, inverted files, and so on were quickly devised but these were relatively complex to implement and demanded expert judgment to select the best method for a particular task (see Figure 1).

IDS was intended to substantially solve these two problems, so that applications could be integrated to share data files and ordinary programmers could effectively develop random access applications using high-level languages. Bachman designed IDS to meet the needs of an integrated systems project called MIACS, for Manufacturing Information and Control System. General Electric had many factories spread over its various divisions, and could not produce and support a different integrated manufacturing system for each one. Furthermore, it was entering the computer business, and its managers recognized that a flexible and generic integrated system based on disk storage would be a powerful tool in selling its machines to other companies. A prototype version of MIACS was being built and tested on the firm's Low Voltage Switch Gear department by a group of systems-minded staff specialists.

### Was IDS a Database Management System?

By interposing itself between application programs and the disk files in which they stored data, IDS carried out

**IDS was designed to be used with a high-level programming language.**

what we still consider the core task of a database management system. Programs could not manipulate data files directly, instead making calls to IDS so that it would perform the data operations on their behalf.

Like modern database management systems, IDS explicitly stored and manipulated metadata about the records and their relationships, rather than expecting each application program to understand and respect the format of every data file it worked with. It enforced relationships between different record types, and would protect database integrity. Database designers specified record clusters, linked list sequencing, indexes, and other details of record organization to boost performance based on expected usage patterns. However, the first versions did not include a formal data description language. Instead of being defined through textual commands the metadata was punched onto specially formatted input cards. A special command told IDS to read and apply this information. New elements could be added without deleting existing records. Each data manipulation command contained a reference to the appropriate element in the metadata.

IDS was designed to be used with a high-level programming language. In the initial prototype version, operational in early 1963, this was General Electric's own GECOM language, though performance and memory concerns drove a shift to assembly language for the application programming in a higher performance version completed in 1964. Calls to IDS operations such as store, retrieve, modify, and delete were evaluated at runtime against embedded metadata. As high-level languages matured and memory grew less scarce, later versions of IDS worked with application programs written in COBOL.

This provided a measure of what is now called data independence for programs. If a file was restructured to add fields or modify their length then the programs using it would continue to work properly. Files could be moved around and records reorganized without rewriting application programs. That made running different application programs against the same database much more feasible. IDS also in-

cluded its own system of paging data in and out of memory, to create a virtual memory capability transparent to the application programmer.

The concept of transactions is fundamental to modern database management systems. Programmers specify that a series of interconnected updates must take place together, so that if one fails or is undone they all are. IDS was also transaction oriented, though not in exactly the same sense. Bachman devised an innovative transaction processing system, which he called the Problem Controller. The Problem Controller and IDS were loaded when the computer was booted. The Problem Controller and IDS occupied 4,000 words of memory. They took control of the entire computer, which might have only 8,000 words of memory. The residual area in memory

was used for paging buffers by IDS's virtual memory manager.

Requests from users to process particular transactions were read from "problem control records" stored and retrieved by IDS in the same manner as application data records. Transactions could be simple, or contain a batch of data cards to be processed. The Problem Controller processed one transaction at a time by executing the designated application program. It worked its way through the queue of transaction requests, choosing the highest priority outstanding job and refreshing the queue from the card reader after each transaction was finished.

The Problem Controller did not appear in later versions of IDS but did provide a basis for an early online transaction processing system. By 1965 an expanded version of the Problem

Controller was built and installed at Weyerhaeuser, on a computer hooked up to a national Teletype network. The system serviced remote users at their Teletypes without any intervention needed by local operators. Requests to process order entry, inventory management, invoicing, and other business transactions were processed automatically by the Problem Controller and application programs.

Bachman's original version of IDS lacked a backup and recovery system, a key feature of later database management systems. This was added in 1964 by the International General Electric team that produced and operated the first production installation of IDS. A recovery and restart magnetic tape logged each new transaction as it was started and captured database pages "before" and "after" they were modified by the transaction, so that the database could be restored to a prior consistent state if something went wrong before the transaction was completed. The same tape also served as a backup of all changes written to the disk in case there was a disk failure since the last full database backup.

The first packaged versions of IDS did lack some features later viewed as essential for database management systems. One was the idea that specific users could be granted or denied access to particular parts of the database. This omission was related to another limitation: IDS databases could be queried or modified only by writing and executing programs in which IDS calls were included. There was no capability to specify "ad hoc" reports or run one-off queries without having to write a program.<sup>a</sup> These capabilities did exist during the 1960s in report generator systems (such as 9PAC and MARK IV) and in online interactive data management systems (such as TDMS) but these packages were generally seen as

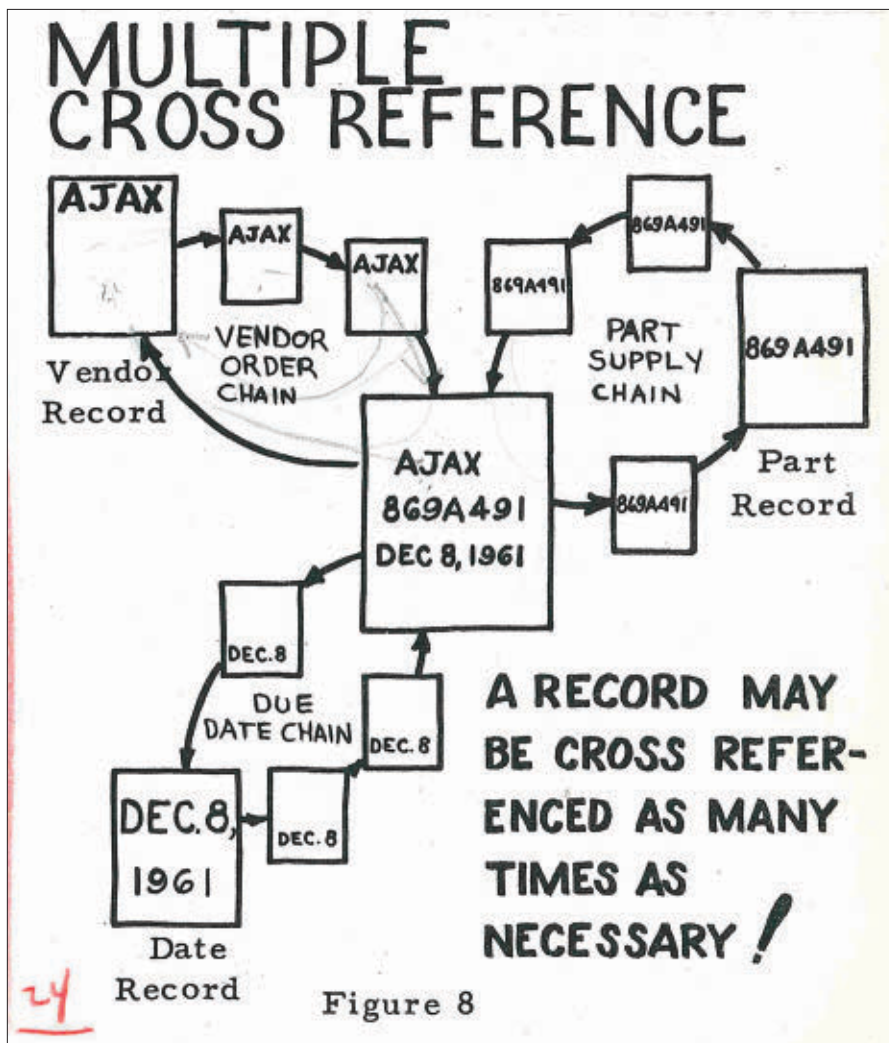


Figure 8

Figure 2. This drawing, from the 1962 presentation "IDS: The Information Processing Machine We Need," shows the use of chains to connect records. The programmer looped through GET NEXT commands to navigate between related records until an end-of-set condition is detected.

<sup>a</sup> On reading this observation, Bachman noted "IDS came into use long before the notion of online, interactive users came into vogue. There is no record of anyone writing an IDS transaction processing application program that processed transactions that specified a query or report and returned the desired output. However, the capability of IDS and the Problem Controller to handle such a query or report specifying transaction programs was clearly available. A missed opportunity!"

## IDS was a strong product, in many respects more advanced than IBM's competing IMS that appeared several years later.

a separate class of software from database management systems. By the 1970s report generation packages, still widely used, included optional modules to interface with data stored in database management systems.

### IDS and CODASYL

After Bachman handed IDS over to a different team within General Electric in 1964 it was made available as a documented and supported software package for the company's 200-series computers. In those days software packages from computer manufacturers were paid for by hardware sales and given to customers without an additional charge. Later versions supported its 400- and 600-series systems. New versions followed in the 1970s after Honeywell bought out General Electric's computer business. IDS was a strong product, in many respects more advanced than IBM's competing IMS that appeared several years later. However, IBM machines so dominated the industry that software from other manufacturers was doomed to relative obscurity.

During the late 1960s the ideas Bachman created for IDS were taken up by the Database Task Group of CODASYL, a standards body for the data processing industry best known for its creation and promotion of the COBOL language. Its initial report, issued in 1969, drew heavily on IDS in defining a proposed standard for database management systems, in part thanks to Bachman's own service on the committee.<sup>4</sup> The report documented foundational concepts and vocabulary such as data definition language, data manipulation language, schemas, data

independence, and program independence. It went beyond early versions of IDS by adding security features, including "privacy locks" and "sub-schemas," roughly equivalent to views in modern systems, so that particular programs could be constrained to work with defined subsets of the database.

CODASYL's definition of the architecture of a database management system and its core capabilities were quite close to that included in textbooks to this day. In particular, it suggested that a database management system should support online, interactive applications as well as batch-driven applications and have separate interfaces. In retrospect, the committee's work, and a related effort by CODASYL's Systems Committee to evaluate existing systems within the new framework,<sup>5</sup> were significant primarily for formulating and spreading the concept of a "database management system."

Although IBM itself refused to support the CODASYL approach many other computer vendors endorsed the committee's recommendations and eventually produced systems incorporating these features. The most successful CODASYL system, IDMS, came from an independent software company. It began as a port of IDS to IBM's dominant System/360 mainframe platform.<sup>6</sup>

### The Legacy of IDS

IDS and CODASYL systems did not use the relational data model, formulated years later by Ted Codd, which underlies today's dominant SQL database management systems. Instead it introduced what would later be called the "network data model." This encoded relationships between different kinds of records as a graph, rather than the strict hierarchy enforced by tape systems and some other software packages of the 1960s such as IBM's later and widely used IMS. The network data model was widely used during the

<sup>b</sup> The importance of the database management system to the emerging packaged software industry is a major theme in M. Campbell-Kelly, *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry*. MIT Press, Cambridge, MA, 2003 and is explored in detail in T.J. Bergin and T. Haigh, "The Commercialization of Database Management Systems, 1969–1983." *IEEE Annals of the History of Computing* 31, 4 (Oct.–Dec. 2009), 26–41.

## Calendar of Events

### July 4–8

MobiHoc'16: The 17<sup>th</sup> ACM International Symposium on Mobile Ad Hoc Networking and Computing, Paderborn, Germany, Sponsored: ACM/SIG, Contact: Falko Dressler Email: dressler@ccs-labs.org

### July 5–8

LICS '16: 31<sup>st</sup> Annual ACM/IEEE Symposium on Logic in Computer Science, New York, NY, Contact: Eric Koskinen Email: eric.koskinen@yale.edu

### July 9–13

ITiCSE '16: Innovation and Technology in Computer Science Education Conference 2016, Arequipa, Peru, Sponsored: ACM/SIG, Contact: Alison Clear Email: aclear@eit.ac.nz

### July 10–13

HT '16: 27<sup>th</sup> ACM Conference on Hypertext and Social Media, Halifax, NS, Canada, Sponsored: ACM/SIG, Contact: Eelco Herder, Email: herder@l3s.de

### July 11–13

SPAA '16: 28<sup>th</sup> ACM Symposium on Parallelism in Algorithms and Architectures, Pacific Grove, CA, Co-Sponsored: ACM/SIG

### July 11–13

SCA '16: The ACM SIGGRAPH/Eurographics Symposium on Computer Animation, Zurich, Switzerland, Sponsored: ACM/SIG, Contact: Matthias Teschner Email: teschner@informatik.uni-freiburg.de

### July 13–17

UMAP '16: User Modeling, Adaptation and Personalization Conference, Halifax, NS, Canada, Co-Sponsored: ACM/SIG, Contact: Julita Vassileva, Email: jiv@cs.usask.ca

1970s and 1980s, and commercial database management systems based on this approach were among the most successful products of the mushrooming packaged software industry.

Bachman spoke memorably in his 1973 Turing Award lecture of the “Programmer as Navigator,” charting a path through the database from one record to another.<sup>3</sup> The network approach used in IDS required programmers to work with one record at a time. Performing the same operation on multiple records meant retrieving a record, processing and if necessary updating it, and then moving on to the next record of interest to repeat the process. For some tasks this made programs longer and more cumbersome than the equivalent in a relational system, where a task such as deleting all records more than a year old or adding 10% to the sales price of every item could be performed with a single command.

IDS and other network systems encoded what we now think of as the “joins” between different kinds of records as part of the database structure rather than specifying them in each query and rebuilding them when the query is processed (see Figure 2). Bachman introduced a data structure diagramming, often called the “Bachman diagram” to describe these relationships.<sup>c</sup> Hardcoding the relationships between record sets made IDS much less flexible than later relational systems, but also much simpler to implement and more efficient for routine operations.

IDS was a useful and practical tool for business use from the mid-1960s, while relational systems were not commercially significant until the early 1980s. Relational systems did not become feasible until computers were orders of magnitude more powerful than they had been in 1963 and some extremely challenging implementation issues had been overcome by pioneers such as IBM’s System R group and Berkeley’s INGRES team. Even after relational systems were commer-

## IDS was a useful and practical tool for business use from the mid-1960s, while relational systems were not commercially available until the early 1980s.

cialized the two approaches were seen for some time as complementary, with network systems used for high-performance transaction-processing systems handling routine operations on large numbers of records (for example, credit card transaction processing or customer billing) and relational systems best suited for “decision support” analytical data crunching. IDMS, the successor to IDS, underpins some very large mainframe applications and is still being supported and enhanced by its current owner Computer Associates, most recently with release 18.5 in 2014. However it, and other database management systems based on Bachman’s network data model, have long since been superseded for new applications and for mainstream computing needs.

Although by any standard a successful innovator, Bachman does not fit neatly into the “hackers, geniuses, and geeks” framework favored by Walter Isaacson. During his long career Bachman had also founded a public company, played a leading role in formulating the OSI seven-layer model for data communications, and pioneered online transaction processing. In 2014, he visited the White House to receive from President Obama a National Medal of Technology and Innovation in recognition of his “fundamental inventions in database management, transaction processing, and software engineering.”<sup>d</sup> Bachman

sees himself above all as an engineer, retaining a professional engineer’s zest for the elegant solution of difficult problems and faith in the power of careful and rational analysis. As he wrote in a note at the end of the transcript of an oral history interview I conducted with him in 2004, “My work has been my play.”<sup>1</sup>

When database specialists look at IDS today they immediately see its limitations compared to modern systems. Its strengths are more difficult to recognize, because its huge influence on the nascent software industry meant that much of what was revolutionary about it in 1963 was soon taken for granted. Without IDS, or Bachman’s tireless championing of the ideas it contained, the very concept of a “database management system” might never have taken root. IDS did more than any other single piece of software to broaden the range of business problems to which computers could usefully be applied and so to usher in today’s “digital” world where every administrative transaction is realized through a flurry of database queries and updates rather than by completing, routing, and filing in triplicate a set of paper forms. **C**

### References

1. Bachman, C.W. Oral history interview by Thomas Haigh September 25–26, 2004, Tucson, AZ. ACM Oral History Interviews collection. ACM Digital Library, <http://dl.acm.org/citation.cfm?id=1141882>.
2. Bachman, C.W. The origin of the integrated data store (IDS): The first direct-access DBMS. *IEEE Annals of the History of Computing* 31, 4 (Oct.–Dec. 2009), 42–54.
3. Bachman, C.W. The programmer as navigator. *Commun. ACM* 16, 11 (Nov. 1973), 653–658.
4. CODASYL Data Base Task Group. *CODASYL Data Base Task Group: October 1969 Report*.
5. CODASYL Systems Committee. *Survey of Generalized Data Base Management Systems*, May 1969. Association for Computing Machinery, New York, 1969.
6. Haigh, T. Charles W. Bachman: Database software pioneer. *IEEE Annals of the History of Computing* 33, 4 (Oct.–Dec. 2011), 70–80.
7. Haigh, T. How data got its base: Generalized information storage software in the 1950s and 60s. *IEEE Annals of the History of Computing* 31, 4 (Oct.–Dec. 2009), 6–25.
8. Haigh, T. Inventing information systems: The systems men and the computer, 1950–1968. *Business History Review* 75, 1 (Spring 2001), 15–61.

**Thomas Haigh** (thomas.haigh@gmail.com) is a visiting professor at Siegen University, an associate professor of information studies at the University of Wisconsin—Milwaukee, and immediate past chair of the SIGCIS group for historians of computing.

<sup>c</sup> C.W. Bachman, “Data Structure Diagrams,” *Data Base 1*, 2 (Summer 1969), 4–10 was very influential in spreading the idea of data structure diagrams, but internal GE documents make clear he was using a similar technique as early as 1962.

<sup>d</sup> The 2012 medals were presented at the White House in November 2014.

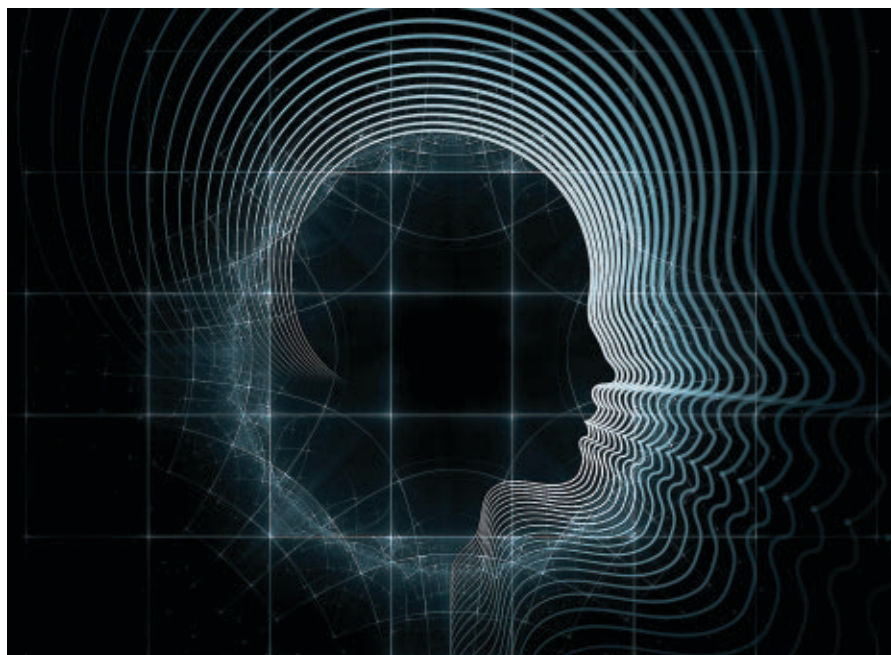
# Computing Ethics

## Big Data Analytics and Revision of the Common Rule

*Reconsidering traditional research ethics given the emergence of big data analytics.*

**B**IG DATA<sup>a</sup> is a major technical advance in terms of computing expense, speed, and capacity. But it is also an epistemic shift wherein data is seen as infinitely networkable, indefinitely reusable, and significantly divorced from the context of collection.<sup>1,7</sup> The statutory definitions of “human subjects” and “research” are not easily applicable to big data research involving sensitive human data. Many of the familiar norms and regulations of research ethics formulated to prior paradigms of research risks and harms, and thus the formal triggers for ethics review are miscalibrated. We need to reevaluate longstanding assumptions of research ethics in light of the emergence of “big data” analytics.<sup>6,10,13</sup>

The U.S. Department of Health and Human Services (HHS) released a Notice of Proposed Rule-Making (NPRM) in September 2015 regarding proposed major revisions (the first in three decades) to the research ethics regulations known as the Common Rule.<sup>a</sup> The proposed changes grapple with the consequences of big data, such as informed consent for biobanking and universal standards for privacy protection. The Common Rule



does not apply to industry research, and some big data science in universities might not fall under its purview, but the Common Rule addresses the burgeoning uses of big data by setting the tone and agenda for research ethics in many spheres.

The NSF-supported Council for Big Data, Ethics and Society<sup>b</sup> has focused on the consequences these proposed changes for big data, including data science and analytics.<sup>9</sup> There is reason

for concern that the rules as drafted in NPRM may muddle attempts to identify and promulgate responsible data science research practices.

### Is Biomedicine the Ethical Baseline?

The Common Rule was instituted in 1981. It mandates federally funded research projects involving human subjects to receive prior, independent ethics review before commencing. Most projects go through Institutional Review Boards (IRB)<sup>3</sup> responsible for

<sup>a</sup> So named for its common application across signatory federal agencies.

<sup>b</sup> See <http://bdes.datasociety.net>

## INTERACTIONS



ACM's *Interactions* magazine explores critical relationships between people and technology, showcasing emerging innovations and industry leaders from around the world across important applications of design thinking and the broadening field of interaction design.

Our readers represent a growing community of practice that is of increasing and vital global importance.



To learn more about us, visit our award-winning website  
<http://interactions.acm.org>

Follow us on Facebook and Twitter



To subscribe:  
<http://www.acm.org/subscribe>

Association for  
Computing Machinery



researchers' due diligence in identifying and ameliorating potential physiological, psychological, and informational harms to human subjects. The Common Rule grew out of a regulatory process initiated by the 1974 National Research Act, a response to public scandals in medical and psychological research, including the Nuremberg Doctors Trial, the Tuskegee syphilis study, and Milgram experiment on obedience to authority figures. The Act led to a commission on human-subjects research ethics that produced the Belmont Report (1979). The Belmont authors insisted that certain core philosophical principles must guide research involving human subjects: respect for persons, beneficence, and justice. The HHS developed the specific regulations in the Common Rule as an instantiation of those principles.<sup>12</sup>

Importantly, the Belmont authors understood that not all activities that produce knowledge or intervene in human lives are "research," and not all research about humans is sensitive or personal enough to be about "human subjects." To delimit "human-subjects research" within biomedicine, the Belmont commission considered "the boundaries between biomedical and behavioral research and the accepted and routine practice of medicine."<sup>12</sup> This boundary reflects the ethical difficulties posed by unique social roles of physician-researchers who are responsible for *both* patient health and societal well-being fostered by research knowledge. This unique role creates ethical dilemmas that are often not reflected in other disciplines. *Research* defined by the Belmont Report is, "an activity designed to test an hypothesis, permit conclusions to be drawn, and thereby to develop or contribute to generalizable knowledge." *Practice* is, "interventions that are designed solely to enhance the well-being of an individual patient or client and that have a reasonable expectation of success."<sup>12</sup>

Not surprisingly, the first draft of the Common Rule came under attack from social scientists for lumping together all forms of human-subjects research under a single set of regulations that reflect the peculiarities of biomedical research.<sup>2</sup> Not all research has the same risks and norms as biomedicine. A single set of rules might snuff out legiti-

mate lines of inquiry, even those dedicated to social justice ends. The HHS responded by creating an "Exempt" category that allowed human-subjects research with minimal risk to receive expedited ethics review. Nevertheless, there has remained a low-simmering conflict between social scientists and IRBs. This sets the stage for debates over regulating research involving big data. For example, in her analysis of the Facebook emotional contagion controversy, Michelle Meyer argues that big data research, especially algorithmic A/B testing without clear temporal boundaries or hypotheses, clouds the distinction between practice and research.<sup>8,11</sup> Jacob Metcalf and Kate Crawford agree this mismatch exists, but argue that core norms of human-subjects research regulations can still be applied to big data research.<sup>10</sup>

### Big Data and the Common Rule Revisions

The Common Rule has typically not been applied to the core disciplines of big data (computing, mathematics, and statistics) because these disciplines are assumed to be conducting research on systems, not people. Yet big data has brought these disciplines into much closer intellectual and economic contact with sensitive human data, opening discussion about how the Common Rule applies. The assumptions behind NPRM leaving big data science out of its purview are empirically suspect.

### Excluded—A New Category

Complaints about inconsistent application of the *exempt* category have prompted HHS to propose a new category of *excluded* that would automatically receive no ethical review due to inherently "*low risk*" to human subjects (§\_\_.101(b)(2)). Of particular interest is exclusion of:

- research involving the collection or study of information that has been or will be acquired solely for non-research activities, **or**
- was acquired for research studies other than the proposed research study when the sources are publicly available, **or**
- the information is recorded by the investigator in such a manner that human subjects cannot be identified, directly or through identifiers linked to

## The contentious history of the Common Rule is due in part to its influence on the tone and agenda of research ethics even outside of its formal purview.

the subjects, the investigator does not contact the subjects, and the investigator will not re-identify subjects or otherwise conduct an analysis that could lead to creating individually identifiable private information. (§\_\_101(b)(2)(ii))<sup>4</sup>

These types of research in the context of big data present different risk profiles depending on the *contents* and *what is done* with the dataset. Yet they are *excluded* based on the assumption that their *status* (public, private, pre-existing, de-identified, and so forth) is an adequate proxy for risk. The proposal to create an excluded category is driven by frustrations of social and other scientists who use data already in the public sphere or in the hands of corporations to whom users turn over mountains of useful data. Notably, social scientists have pushed to define “public datasets” such that it includes datasets that can be purchased.<sup>2</sup> The power and peril of big data research is that large datasets can theoretically be correlated with other large datasets in novel contexts to produce unforeseeable insights. Algorithms might find unexpected correlations and generate predictions as a possible source of poorly understood harms. Exclusion would eliminate ethical review to address such risks.

*Public* and *private* are used in the NPRM in ways that leave this regulatory gap open. “Public” modifies “datasets,” describing access or availability. “Private” modifies “information” or “data” describing a reasonable subject’s expectations about sensitivity. Yet *publicly available datasets containing private data* are among the most interesting to

researchers and most risky to subjects.

For example, a recent study by Hauge et al.<sup>5</sup> used geographic profiling techniques and public datasets to (allegedly) identify the pseudonymous artist Banksy. The study underwent ethics review, and was (likely) permitted because it used public datasets, despite its intense focus on the private information of individual subjects.<sup>5</sup> This discrepancy is made possible by the anachronistic assumption that any informational harm has already been done by a public dataset. That the NPRM explicitly cites this assumption as a justification to a priori exclude increasingly prominent big data research methods is highly problematic.

Perhaps academic researchers *should* have relaxed access to maintain parity with industry or further scientific knowledge. But the Common Rule should not allow that de facto under the guise of empirically weak claims about the risks posed by public datasets. The Common Rule might rightfully exclude big data research methods from its purview, but it should do so explicitly and not muddle attempts to moderate the risks posed by declaring public data inherently low risk.

### Exempt—An Expanded Category

The NPRM also proposes to expand the Exempt category (minimal review largely conducted through an online portal) to include secondary research using datasets containing identifiable information collected for non-research purposes. All such research would be exempt as long as subjects were given prior notice and the datasets are to be used only in the fashion identified by the requestor (§\_\_104(e)(2)). The NPRM does not propose to set a minimum bar for adequate notice. This can be reasonable given the high standard of informed consent is intended primarily for medical research, and can be an unreasonable burden in the social sciences. However, to default to end user license agreements (EULA) poses too low a bar. Setting new rules for the exempt category should not be a de facto settlement of this open debate. Explicit guidelines and processes for future inquiry and revised regulations are warranted.

### Conclusion

The NPRM improves the Common Rule’s application to big data research, but portions of the NPRM with consequences for big data research rest on dated assumptions. The contentious history of the Common Rule is due in part to its influence on the tone and agenda of research ethics even outside of its formal purview. This rare opportunity for significant revisions should not cement problematic assumptions into the discourse of ethics in big data research. **C**

### References

1. boyd, d. and Crawford, K. Critical questions for big data. *Information, Communication & Society* 15, 5 (2012), 662–679.
2. Committee on Revisions to the Common Rule for the Protection of, Board on Behavioral, Cognitive, and Sensory Sciences, Committee on National Statistics, et al. Proposed Revisions to the Common Rule for the Protection of Human Subjects in the Behavioral and Social Sciences, 2014; <http://www.nap.edu/read/18614/chapter/1>.
3. Department of Health and Human Services Code of Federal Regulations Title 45—Public Welfare, Part 46—Protection of Human Subjects. 45 Code of Federal Regulations 46, 2009; <http://www.hhs.gov/ohrp/humansubjects/guidance/45cfr46.html>.
4. Department of Health and Human Services. Notice of Proposed Rule Making: Federal Policy for the Protection of Human Subjects. Federal Register, 2015; <http://www.gpo.gov/fdsys/pkg/FR-2015-09-08/pdf/2015-21756.pdf>.
5. Hauge, M.V. et al. Tagging Banksy: Using geographic profiling to investigate a modern art mystery. *Journal of Spatial Science* (2016): 1–6.
6. King, J.L. Humans in computing: Growing responsibilities for researchers. *Commun.* 58, 3 (Mar. 2015), 31–33.
7. Kitchin, R. Big data, new epistemologies and paradigm shifts. *Big Data & Society* 1, 1 (2014).
8. Kramer, A., Guillory, J., and Hancock, J. Experimental evidence of massive-scale emotional contagion through social networks. In *Proceedings of the National Academy of Sciences* 111, 24 (2014), 8788–8790.
9. Metcalf, J. Letter on Proposed Changes to the Common Rule. Council for Big Data, Ethics, and Society (2016); <http://bdes.datasociety.net/council-output/letter-on-proposed-changes-to-the-common-rule/>.
10. Metcalf, J. and Crawford, K. Where are human subjects in big data research? The emerging ethics divide. *Big Data & Society* 3, 1 (2016), 1–14.
11. Meyer, M.N. Two cheers for corporate experimentation: The a/b illusion and the virtues of data-driven innovation. *Colorado Technology Law Journal* 13, 273 (2015).
12. National Commission for the Protection of Human Subjects, of Biomedical and Behavioral Research and The National Commission for the Protection of Human Subjects (1979) The Belmont Report: Ethical Principles and Guidelines for the Protection of Human Subjects of Research; <http://www.hhs.gov/ohrp/humansubjects/guidance/belmont.html>.
13. Zwitter, A. Big data ethics. *Big Data & Society* 1, 2 (2014).

**Jacob Metcalf** ([jake.metcalf@gmail.com](mailto:jake.metcalf@gmail.com)) is a Researcher at the Data & Society Research Institute, and Founding Partner at the ethics consulting firm Ethical Resolve.

This work is supported in part by National Science Foundation award #1413864. See J. Metcalf “Letter on Proposed Changes to the Common Rule. Council for Big Data, Ethics, and Society (2016)”<sup>9</sup> for the public comment on revisions to the Common Rule published collectively by the Council for Big Data, Ethics and Society. This column represents only the author’s opinion.

Copyright held by author.

# Viewpoint

## Turing's Red Flag

*A proposal for a law to prevent artificial intelligence systems from being mistaken for humans.*

**M**OVIES CAN BE a good place to see what the future looks like. According to Robert Wallace, a retired director of the CIA's Office of Technical Service: "... When a new James Bond movie was released, we always got calls asking, 'Do you have one of those?' If I answered 'no', the next question was, 'How long will it take you to make it?' Folks didn't care about the laws of physics or that Q was an actor in a fictional series—his character and inventiveness pushed our imagination ..."<sup>3</sup> As an example, the CIA successfully copied the shoe-mounted spring-loaded and poison-tipped knife in *From Russia With Love*. It's interesting to speculate on what else Bond movies may have led to being invented.

For this reason, I have been considering what movies predict about the future of artificial intelligence (AI). One theme that emerges in several science fiction movies is that of an AI mistaken for human. In the classic movie *Blade Runner*, Rick Deckard (Harrison Ford) tracks down and destroys replicants that have escaped and are visually indistinguishable from humans. Tantalizingly, the film leaves open the question of whether Rick Deckard is himself a replicant. More recently, the movie *Ex Machina* centers around a type of Turing Test in which the robot Ava tries to be convincingly human enough to trick someone into helping her escape. And in *Metropolis*, one of the very first science fiction movies ever, a robot disguises itself as the woman Maria and thereby causes the workers to revolt.



**The 19<sup>th</sup>-century U.K. Locomotive Act, also known as the Red Flag Act, required motorized vehicles to be preceded by a person waving a red flag to signal the oncoming danger.**

It thus seems likely that sometime in the future we will have to deal with the impact of AI's being mistaken for humans. In fact, it could be argued that this future is already here. Joseph Weizenbaum proposed ELIZA as a "parody" of a psychotherapist and first described the program in the pages of *Communications* in 1966.<sup>4</sup> However, his secretary famously asked to be left alone so she could talk in private to the chatterbot. More recently a number of different chatterbots have fooled judges in the annual Loebner prize, a version of the Turing Test.

Alan Turing, one of the fathers of artificial intelligence, predicted in 1950 that computers would be mistaken for humans in around 50 years.<sup>2</sup> We may be running a little late on this prediction. Nevertheless the test Alan Turing proposed in the very same paper that contains his 50-year prediction remains the best known test for AI (even if there are efforts under way to update and refine his test). Let us not forget that the Turing Test is all about an AI passing itself off as a human. Even if you are not a fan of the Turing Test,

it nevertheless has placed the idea of computers emulating humans firmly in our consciousness.

As any lover of Shakespeare knows, there are many dangers awaiting us when we try to disguise our identity. What happens if the AI impersonates someone we trust? Perhaps they will be able to trick us to do their bidding. What if we suppose they have human-level capabilities but they can only act at a sub-human level? Accidents might quickly follow. What happens if we develop a social attachment to the AI? Or worse still, what if we fall in love with them? There is a minefield of problems awaiting us here.

This is not the first time in history that a technology has come along that might disrupt and endanger our lives. Concerned about the impact of motor vehicles on public safety, the U.K. parliament passed the Locomotive Act in 1865. This required a person to walk in front of any motorized vehicle with a red flag to signal the oncoming danger. Of course, public safety wasn't the only motivation for this law as the railways profited from restricting motor vehicles in this way. Indeed, the law clearly restricted the use of motor vehicles to a greater extent than safety alone required. And this was a bad thing. Nevertheless, the sentiment was a good one: until society had adjusted to the arrival of a new technology, the public had a right to be forewarned of potential dangers.

Interestingly, this red flag law was withdrawn three decades later in 1896 when the speed limit was raised to 14mph (approximately 23kph). Coincidentally, the first speeding offense, as well as the first British motoring fatality, the unlucky pedestrian Bridget Driscoll also occurred in that same year. And road accidents have quickly escalated from then on. By 1926, the first year for which records are available, there were 134,000 cases of serious injury, yet there were only 1,715,421 vehicles on the roads of Great Britain. That is one serious injury each year for every 13 vehicles on the road. And a century later, thousands still die on our roads every year.

Inspired by such historical precedents, I propose that a law be enacted to prevent AI systems from being mistaken for humans. In recognition of

## There are many reasons we don't want computers to be intentionally or unintentionally fooling us.

Alan Turing's seminal contributions to this area, I am calling this the Turing Red Flag law.

**Turing Red Flag law:** *An autonomous system should be designed so that it is unlikely to be mistaken for anything besides an autonomous system, and should identify itself at the start of any interaction with another agent.*

Let me be clear. This is not the law itself but a summary of its intent. Any law will have to be much longer and much more precise in its scope. Legal experts as well as technologists will be needed to draft such a law. The actual wording will need to be carefully crafted, and the terms properly defined. It will, for instance, require a precise definition of autonomous system. For now, we will consider any system that has some sort of freedom to act independently. Think, for instance, self-driving car. Though such a car does not choose its end destination, it nevertheless does independently decide on the actual way to reach that given end destination. I would also expect that, as is often the case in such matters, the exact definitions will be left to the last moment to leave bargaining room to get any law into force.

There are two parts to this proposed law. The first part of the law states that an autonomous system should not be designed to act in a way that it is likely to be mistaken there is a human in the loop. Of course, it is not impossible to think of some situations where it might be beneficial for an autonomous system to be mistaken for something other than an autonomous system. An AI system pretending to be human might, for example, create more engaging interactive fiction. More controversially, robots pretending to be human

might make better caregivers and companions for the elderly. However, there are many more reasons we don't want computers to be intentionally or unintentionally fooling us. Hollywood provides lots of examples of the dangers awaiting us here. Such a law would, of course, cause problems in running any sort of Turing Test. However, I expect that the current discussion about replacements for the Turing Test will eventually move from tests for AI based on deception to tests that quantify explicit skills and intelligence. Some related legislation has been put into law for guns. In particular, former California Governor Schwarzenegger signed legislation in September 2004 that prohibits the public display of toy guns in California unless they are clear or painted a bright color to differentiate them from real firearms. The purpose of this law is to prevent police officers mistaking toy guns for real ones.

The second part of the law states that autonomous systems need to identify themselves at the start of any interaction with another agent. Note that this other agent might even be another AI. This is intentional. If you send your AI bot out to negotiate the purchase of a new car, you want the bot also to know whether it is dealing with a dealer bot or a person. You wouldn't want the dealer bot to be able to pretend to be a human just because it was interacting with your bot. The second part of the law is designed to reduce the chance that autonomous systems are accidentally mistaken for what they are not.

Consider four up-and-coming areas where this law might have bite. First, consider autonomous vehicles. I find it a real oversight that the first piece of legislation that permits autonomous vehicles on roads, the AB 511 act in Nevada, says nothing at all about such vehicles being identified to other road users as autonomous. A Turing Red Flag law, on the other hand, would require an autonomous vehicle identify itself as autonomously driven both to human drivers and to other autonomous vehicles. There are many situations where it could be important to know that another road vehicle is being driven autonomously. For example, when a light changes we can suppose that an autonomous vehicle approaching the light will

# COMMUNICATIONS APPS

Access the latest issue, past issues, **BLOG@CACM**, News, and more.



Available for iPad, iPhone, and Android



Available for iOS, Android, and Windows

<http://cacm.acm.org/about-communications/mobile-apps>



Association for Computing Machinery

indeed stop, and so save us from having to brake hard to avoid an accident. As a second example, if an autonomous car is driving in front of us in fog, we can suppose it can see a clear road ahead using its radar. For this reason, we do not have to leave a larger gap in case it has to brake suddenly. As a third example, at a four-way intersection, we can suppose an autonomous car will not aggressively pull out when it does not have right of way. And as a fourth and final example, if an autonomous car arrives at a diversion, we might expect it to drive more slowly as it tries to work out where the road is now going.

How should an autonomous vehicle identify itself? I don't suppose this should be with a person walking in front with a red flag. This was too restrictive even back in 1865. Autonomous vehicles might have to carry distinctive plates, just like we require learner drivers to identify themselves on the roads today. Or autonomous vehicles might have to display a magenta flashing light whenever they are being operated autonomously. In addition, autonomous vehicles should broadcast their location, velocity, and autonomy to neighboring vehicles.

In June 2015, Reuters reported that two autonomous cars, one from Google and the other from Delphi Automotive Plc, nearly had an accident on a Silicon Valley street. The Google car apparently cut off the Delphi car as it was about to perform a lane change. The Delphi car then had to take "appropriate action" to avoid an accident. Clearly, it would help prevent such incidents if autonomous vehicles were required to broadcast their location and intentions. Of course, adequate safeguards will also need to be put in place that such broadcasting does not compromise the privacy of their human occupants.

Once autonomous vehicles are commonplace, other motorists will expect to meet autonomous vehicles. But before this time, it will be important to know that the oncoming vehicle is somewhat special and may behave differently to a human driver. In the past, it was less necessary for autonomous vehicles to identify themselves. There were few of them and most looked a little odd. The presence of large rotating LIDAR sensors on their roofs has

tended to give the game away. However, now that the technology is shrinking and autonomous cars are starting to look more normal, it will become increasingly difficult to know when a car is being driven autonomously. There is also a reverse argument. Suppose autonomous vehicles are (or become) better drivers than humans. Then just like today when we demand less experienced and capable learner drivers are identified, we might demand that human drivers are identified apart from autonomous drivers. A Turing Red Flag law will achieve this.

Let's move on to the second area where a Turing Red Flag law might start having impact shortly. Consider virtual assistants like Apple's Siri or Microsoft's Cortana. Here is a recent conversation I had with Siri:

**Siri.** What can I help you with?

**Q.** Are you a computer?

**Siri.** I can neither confirm nor deny my current existential status.

**Q.** Are you a human?

**Siri.** I'm an assistant. That's all that matters.

**Q.** Are you an AI?

**Siri.** That's a rather personal question.

Based on conversations like these, it would appear that Siri is coming close to violating this proposed Turing Red Flag law. It begins its conversations without identifying itself as a computer, and it answers in a way that, depending on your sense of humor, might deceive. At least, in a few years time, when the dialogue is likely more sophisticated, you can imagine being deceived. Of course, few if any people are currently deceived into believing that Siri is human. It would only take a couple of questions for Siri to reveal that it is not human. Nevertheless, it is a dangerous precedent to have technology like this in everyday use on millions of smartphones pretending, albeit poorly, to be human.

There are also several more trusting groups that could already be deceived. My five-year-old daughter has a doll that uses a Bluetooth connection to Siri to answer general questions. I am not so sure she fully appreciates that it is just a smartphone doing all the clever work here. Another troubling group are patients with Alzheimer's disease and other forms of dementia. Paro is a cuddly robot seal that has been trialed as

therapeutic tool to help such patients. Again, some people find it troubling that a robot seal can be mistaken for real. Imagine then how much more troubling society is going to find it when such patients mistake AI systems for humans?

Let's move onto a third example, online poker. This is a multibillion-dollar industry so it is possible to say that the stakes are high. Most, if not all, online poker sites already ban computer bots from playing. Bots have a number of advantages, certainly over weaker players. They never tire. They can compute odds very accurately. They can track historical play very accurately. Of course, in the current state of the art, they also have disadvantages such as understanding the psychology of their opponents. Nevertheless, in the interest of fairness, I suspect most human poker players would prefer to know if any of their opponents was not human. A similar argument could be made for other online computer games. You might want to know if you're being "killed" easily because your opponent is a computer bot with lightning-fast reflexes.

I conclude with a fourth example: computer-generated text. Associated Press now generates most of its U.S. corporate earnings reports using a computer program developed by Automated Insights.<sup>1</sup> A narrow interpretation might rule such computer-generated text outside the scope of a Turing Red Flag law. Text-generation algorithms are typically not autonomous. Indeed, they are typically not interactive. However, if we consider a longer time scale, then such algorithms are interacting in some way with the real

**I suspect most human poker players would prefer to know if any of their opponents was not human.**

world, and they may well be mistaken for human-generated text. Personally, I would prefer to know whether I was reading text written by human or computer—it is likely to impact my emotional engagement with the text. But I fully accept that we are now in a grey area. You might be happy for automatically generated tables of stock prices and weather maps to be unidentified as computer generated, but perhaps you do want match reports to be identified as such? What if the commentary on the TV show covering the World Cup Final is not Messi, one of the best footballers ever, but a computer that just happens to sound like Messi? And should you be informed if the beautiful piano music being played on the radio is composed by Chopin or by a computer in the style of Chopin? These examples illustrate that we still have some way to go working out where to draw the line with any Turing Red Flag law. But I would argue, there is a line to be drawn somewhere here.

There are several arguments that can be raised against a Turing Red Flag law. One argument is that it's way too early to be worrying about this problem now. Indeed, by flagging this problem today, we're just adding to the hype around AI systems breaking bad. There are several reasons why I discount this argument. First, autonomous vehicles are likely only a few years away. In June 2011, Nevada's Governor signed into law AB 511, the first legislation anywhere in the world that explicitly permits autonomous vehicles. As I mentioned earlier, I find it surprising that the bill says nothing about the need for autonomous vehicles to identify themselves. In Germany, autonomous vehicles are currently prohibited based on the 1968 Vienna Convention on Road Traffic to which Germany and 72 other countries follow. However, the German transport minister formed a committee in February 2015 to draw up the legal framework that would make autonomous vehicles permissible on German roads. This committee has been asked to present a draft of the key points in such a framework before the Frankfurt car fair in September 2015. We may therefore already be running late to ensure autonomous vehicles identify themselves on German roads. Second, many of us have already been

fooled by computers. Several years ago a friend asked me how the self-service checkout could recognize different fruit and vegetables. I hypothesized a classification algorithm, based on color and shape. But then my friend pointed out the CCTV display behind me with a human operator doing the classification. The boundary between machine and man is quickly blurring. Even experts in the field can be mistaken. A Turing Red Flag law will help keep this boundary sharp. Third, humans are often quick to assign computers with more capabilities than they actually possess. The last example illustrates this. As another example, I let some students play with an Aibo robot dog, and they quickly started to ascribe the Aibo with emotions and feelings, neither of which the Aibo has. Autonomous systems will be fooling us as human long before they actually are capable to act like humans. Fourth, one of the most dangerous times for any new technological is when the technology is first being adopted, and society has not yet adjusted to it. It may well be, as with motor cars today, society decides to repeal any Turing Red Flag laws once AI systems become the norm. But while they are rare, we might well choose to act a little more cautiously.

In many U.S. states, as well as many countries of the world including Australia, Canada, and Germany, you must be informed if your telephone conversation is about to be recorded. Perhaps in the future it will be routine to hear, "You are about to interact with an AI bot. If you do not wish to do so, please press 1 and a real person will come on the line shortly." ■

#### References

1. Colford, P. A leap forward in quarterly earnings stories. Associated Press blog announcement, 2014; <https://blog.ap.org/announcements/a-leap-forward-in-quarterly-earnings-stories>.
2. Turing, A. Computing machinery and intelligence. *MIND: A Quarterly Review of Psychology and Philosophy* 59, 236 (1950), 433–460.
3. Wallace, R., Melton, H., and Schlesinger, R. *Spycraft: The Secret History of the CIA's Spytechs, from Communism to al-Qaeda*. Dutton Adult, 2008.
4. Weizenbaum, J. Eliza—A computer program for the study of natural language communication between man and machine. *Commun. ACM* 9, 1 (Jan. 1966), 36–45.

**Toby Walsh** ([toby.walsh@nicta.com.au](mailto:toby.walsh@nicta.com.au)) is Professor of Artificial Intelligence at the University of New South Wales, and Data61, Sydney, Australia. He was recently elected a Fellow of the Australian Academy of Sciences.

Copyright held by author.

## Viewpoint

# Inverse Privacy

*Seeking a market-based solution to the problem of a person's unjustified inaccessibility to their private information.*

**C**ALL AN ITEM of your personal information inversely private if some party has access to it but you do not. The provenance of your inversely private information can be totally legitimate. Your interactions with various institutions—employers, municipalities, financial institutions, health providers, police, toll roads operators, grocery chains, and so forth—create numerous items of personal information, for example, shopping receipts and refilled prescriptions. Due to progress in technology, institutions have become much better than you in recording data. As a result, shared data decays into inversely private. More inversely private information is produced when institutions analyze your private data.

Your inversely private information, whether collected or derived, allows institutions to serve you better. But access to that information—especially if it were presented to you in a convenient form—would do you much good. It would allow you to correct possible errors in the data, to have a better idea of your health status and your credit rating, and to identify ways to improve your productivity and quality of life.

In some cases, the inaccessibility of your inversely private information can be justified by the necessity to protect the privacy of other people and to protect the legitimate interests of institutions. We argue that there are numerous scenarios where the chances to hurt other parties by providing you access to your data are negligible. The



inaccessibility of your inversely private information in such safe scenarios is the inverse privacy problem. A good solution to the problem should not only provide you accessibility to your inversely private information but should also make that access convenient.

We analyze the root causes of the inverse privacy problem and discuss a market-based solution for it. We concentrate here on the big picture, leaving many finer points for later analysis.

Some explanations are more natural in a dialogue, and so we include here

some discussions between Quisani, ostensibly a former student of the first author, and the authors.

### Personal Infoset

For brevity, items of information are called *infons*.<sup>11</sup> An infon is *tangible* if it has a material embodiment, for example, written down on a piece of paper or recorded in some database. The same infon (as an abstract item of information) may have distinct material embodiments. Herein we restrict attention to infons that are tangible.

We are interested in scenarios where a person interacts with an institution, for example, a shop, a medical office, a government agency. We say that an infon  $x$  is *personal* to an individual  $P$  if (a)  $x$  is related to an interaction between  $P$  and an institution and (b)  $x$  identifies  $P$ . A typical example of such an infon is a receipt for a credit-card purchase by a customer in a shop.

Define the personal infoset of an individual  $P$  to be the collection of all infons personal to  $P$ . Note that the infoset evolves over time. It acquires new infons. It may also lose some infons. But, because of the tangibility restriction, the infoset is finite at any given moment.

Q: Give me an example of an intangible infon.

A: A fleeting impression that you have of someone who just walked by you.

Q: What about information announced but not recorded at a meeting? One can argue that the collective memory of the participants is a kind of embodiment.

A: Such a case of unrecorded information becomes less and less common. People write notes, write and send email messages, tweet, use their smartphones to make videos, and so forth. Companies tend to tape their meetings. Numerous sensors, such as cameras and microphones, are commonplace and growing in pervasiveness, even in conference rooms. But yes, there are border cases as far as tangibility is concerned. At this stage of our analysis, we abstract them away.

Q: In the shopping receipt example, the receipt may also mention the salesclerk that helped the customer.

A: The clerk represents the shop on the receipt.

Q: But suppose that something went wrong with that particular purchase, the customer complained that the salesclerk misled her, and the shop investigates. In the new context, the person of interest is the salesclerk. The same infon turns out to be personal to more than one individual.

A: This is a good point. The same infon may be personal to more than one individual but we are interested primarily in contexts where the infon in question is personal to one individual.

## A good solution to the problem should not only provide you accessibility to your inversely private information but should also make the access convenient.

### Classification

The personal infoset of an individual  $P$  naturally splits into four buckets.

1. The *directly private bucket* comprises the infons that  $P$  has access to but nobody else does.

2. The *inversely private bucket* comprises the infons that some party has access to but  $P$  does not.

3. The *partially private bucket* comprises the infons that  $P$  has access to and a limited number of other parties do as well.

4. The *public bucket* comprises the infons that are public information.

Q: Why do you call the second bucket “inversely private?”

A: The *Merriam-Webster* dictionary defines “inverse” as “opposite in order, nature, or effect.” The description of bucket 2 is the opposite of that of bucket 1.

Q: As far as I can see, you discuss just two dimensions of privacy: whom a given infon is personal to, and who has access to the infon. The world is more complex, and there are other dimensions to privacy. Consider for example the pictures in the directly private bucket of my infoset that are personal to me only. Some of the pictures are clearly more private than others; there are degrees of privacy.

A: Indeed, we restrict attention to the two dimensions. But this restricted view is informative, and it allows us to carry on our analysis. Recall that we concentrate here on the big picture leaving many finer points for later analysis.

Q: Concerning the public bucket of my infoset, how can public information be personal? Personal and public are the opposites.

A: You may be confusing personal information with its sensitive part. Not every personal infon is sensitive. For example, the name of our president is personal information as well as public.

### Provenance

With time, the personal infoset of an individual acquires new infons. She may create new infons on her own, for example, by making a selfie, by writing down some observation, or by writing down some conclusions she inferred from information available to her.

But the infoset acquires many more new infons due to the interactions of the individual with other parties. The other parties could be people, such as relatives, neighbors, coworkers, clerks, waiters, and medical personnel. They could be institutions, such as employers, banks, Internet providers, brick-and-mortar shops, online shops, and government agencies. The new infons could be factual records, gossip, rumors, or derived information.

The infoset may also lose some infons, especially if they have a unique embodiment. For example, the individual may destroy old letters or delete a selfie without sending it to anybody. Institutions also may lose or delete (embodiments of) infons, but in general, these days, institutions are much better than people in keeping records.

New items of a personal infoset do not necessarily stay in the bucket where they arose. Because of modern superiority of institutional bookkeeping, there is a flow of information from the partial privacy bucket to the inverse privacy bucket—we look into these dynamics next.

### The Rise of Inverse Privacy to Dominance

People have always interacted among themselves, and people have interacted with institutions for a very long time, certainly from the times that ancient governments started to collect taxes. Until recently the capacity of a person to take and keep records was comparable to that of institutions. Yes, the government kept tax records but,



Association for  
Computing Machinery

## ACM Conference Proceedings Now Available via Print-on-Demand!

*Did you know that you can  
now order many popular  
ACM conference proceedings  
via print-on-demand?*

Institutions, libraries and individuals can choose from more than 100 titles on a continually updated list through Amazon, Barnes & Noble, Baker & Taylor, Ingram and NACSCORP: CHI, KDD, Multimedia, SIGIR, SIGCOMM, SIGCSE, SIGMOD/PODS, and many more.

**For available titles and  
ordering info, visit:  
[librarians.acm.org/pod](http://librarians.acm.org/pod)**



by and large, the people knew about their taxes as much as the government did. Traditionally, the partial privacy bucket easily dominated the inverse privacy bucket.

Later on, governments, especially dictatorial governments, could marshal resources to collect information on people; a novelist illustrated this power the best.<sup>13</sup> The most radical change, however, is due to technology introduced in the last 20–30 years. The capacity of public and private institutions to take and keep records became vastly superior to that of a regular person. As a result, the large majority of items in the personal infoset is now generated as inversely or partially private. Often infons start as partially private but then quickly decay into inversely private because the institutions remember it all while the person often hardly remembers that the interaction took place.

For a regular citizen of an advanced society today, the volume of the inverse privacy bucket vastly exceeds that of the partial privacy bucket. Of course it may be simplistic to count bits or even items. A picture of a car has many bits but only so much useful information; even many pictures of the same car may have only so much useful information. It makes more sense to speak about the value of information rather than its volume.

Determining the value of personal information is a difficult problem, particularly because of a gap between what people are willing to pay for keeping an item of information directly private and what they are willing to accept for sharing that same item of information; see Acquisti et al.<sup>1</sup> and its references. Nevertheless, we posit that typically the value of the inverse privacy bucket exceeds that of the partial privacy bucket and grows much faster.

Thus, in advanced societies today, the inverse privacy bucket of a typical personal infoset dominates the whole infoset. We see the dominance of inverse privacy as a problem. In this connection, it is important to understand legal, political, sociological, ethical, technological implications of the inverse privacy domination.

It is worth emphasizing that the main reason that we live now in a

world dominated by inverse privacy is not the invasion of privacy (the tremendous importance of that issue notwithstanding) but the gross disparity in the capability to take and keep records.

### The Inverse Privacy Entitlement Principle

Enterprises have legitimate reason to collect data about their customers; this allows them to serve their customers better. Medical institutions have legitimate reasons to collect data about their patients; this helps them diagnose and treat diseases. Governments have legitimate reason to collect data about their citizens; this helps them address societal problems.

As noted earlier, institutions are much better than individuals in collecting data. So, in the process of all the collection of data about customers, patients, and citizens, partially private data is quickly becoming inversely private. Aside from any surreptitious collection of personal information, this conversion of data from partially private to inversely private is critical to the provenance of inversely private information.

Access to your inversely private infons would allow you to correct possible errors in the data, to have a better idea of your health status and your credit rating, and so on.

From an ethical point of view, it is only fair to give you access to your personal infons. Already the 1973 HEW report<sup>16</sup> advocated that “[t]here must be no personal-data record-keeping systems whose very existence is secret,” and “[t]here must be a way for an individual, to find out what information about him is in a record and how it is used.” And the 1970 Fair Credit Reporting Act (FCRA) stipulated that, subject to various technical exceptions, “[e]very consumer reporting agency shall, upon request, ... clearly and accurately disclose to the consumer” all information in the consumer’s file, the sources of the information, and so on.<sup>6</sup>

Concentrating on the big picture, we ignore technical exceptions here. But we cannot ignore that governments have legitimate security concerns, and businesses have legitimate competition concerns. The 2012 Federal Trade Commission (FTC) report

on “Protecting Consumer Privacy in an Era of Rapid Change” is more nuanced: “Companies should provide reasonable access to the consumer data they maintain; the extent of access should be proportionate to the sensitivity of the data and the nature of its use.”<sup>8</sup> To this end, we posit:

**The Inverse Privacy Entitlement Principle.** *As a rule, individuals are entitled to access their personal infons. There may be exceptions, but each such exception needs to be justified, and the burden of justification is on the proponents of the exception.*

One obvious exception is related to national security. The proponents of that exception, however, would have to justify it. In particular, they would have to justify which parts of national security fall under the exception.

### The Inverse Privacy Problem

We say that an institution *shares back* your personal infons if it gives you access to them. This technical term will make the exposition easier. Institutions may be reluctant to share back personal information, and they may have reasonable justifications: the privacy of other people needs to be protected, there are security concerns, there are competition concerns. But there are numerous *safe scenarios* where the chances are negligible that sharing back your personal infons would violate the privacy of another person or damage the legitimate interests of the information holding institution or any other institution.

The *inverse privacy problem* is the inaccessibility to you of your personal information in such safe scenarios.

Q: Give me examples of safe scenarios.

A: Your favorite supermarket has plentiful data about your shopping there. Do you have that data?

Q: No, I don't.

A: But, in principle you could have. So how can sharing that data with you hurt anybody? Similarly, many other businesses and government institutions have information about you that you could in principle have but in fact you do not. Some institutions share a part of your inversely private information with you but only a part. For example, Fitbit sends you weekly summaries but they have much more information about you.

## As a rule, individuals are entitled to access their personal infons.

Q: As you mentioned earlier, institutions have not only raw data about me but also derived information. I can imagine that some of that derived information may be sensitive.

A: Yes, there may be a part of your inversely private information that is too sensitive to be shared with you. Our position is, however, that the burden of proof is on the information-holding institution.

Q: You use judicial terminology. But who is the judge here?

A: The ultimate judge is society.

Q: Let me raise another point. Enabling me to access my inversely private information makes it easier for intruders to find information about me.

A: This is true. Any technology invented to allow inverse privacy information to be shared back has to be made secure. Communication channels have to be secure, encryption has to be secure, and so forth. Note, however, that today hackers are in a much better position to find your inversely private information about you than you are. Sharing that information with you should improve the situation.

### Going Forward

As we pointed out previously, the inverse privacy problem is not simply the result of ill will of governments or businesses. It is primarily a side effect of technological progress. Technology influences the social norms of privacy.<sup>17</sup> In the case of inverse privacy, technology created the problem, and technology is instrumental in solving it. Here, we argue that the inverse privacy problem can be solved and will be solved. By default we restrict attention to the safe scenarios described previously.

**Social norms.** Individuals would greatly benefit from gaining access to their inversely private infons. They will have a much fuller picture of their

health, their shopping history, places they visited, and so on. Besides, they would have an opportunity to correct possible errors in inversely private infons. To what extent do people understand the great benefits of accessing their inversely private infons? We do not have data on the subject, but one indication appears in Leon et al.<sup>12</sup>: “We asked participants to [t]hink about the ability to view and edit the information that advertising companies know about you. How much do you agree or disagree with the following, showing them six statements. 90% of participants believed (agreed, strongly agreed) they should be given the opportunity to view and edit their profiles. A large percentage wanted to be able to decide what advertising companies can collect about them (85%) and saw benefits in being able to view (79%) and edit profiles (81%). The majority thought that the ability to edit their profiles would provide companies with more accurate data (70%) and allow them to better serve the participants (64%).”

As people realize the benefits at issue more and more, they will demand access to their inversely private infons louder and louder. Indeed, it is easy to underestimate the amount of information that businesses have about their clients. The story of Austrian privacy activist Max Schrems is instructive. “In 2011, Schrems demanded that Facebook give him all the data the company had about him. This is a requirement of European Union (EU) law. Two years later, after a court battle, Facebook sent him a CD with a 1,200-page PDF.”<sup>14</sup>

Social norms will evolve accordingly, toward a broad acceptance of the Inverse Privacy Entitlement Principle described earlier. Institutions should share back personal information as a matter of course. Furthermore, they should do so in a convenient way. Your personal infons should be available to you routinely and easily—just as the photos that you upload to a reputable cloud store. You do not have to file a legal request to obtain them.

The evolving social norms influence the law, and the law helps to shape social norms. Here, for brevity, we restrict attention to the U.S. law. We already quoted the 1970 Fair Credit Reporting Act, the 1973 HEW report, and the 2012

FTC report. Here are some additional laws and FTC reports of relevance:

- A 2000 report of an FTC Advisory Committee on “providing online consumers reasonable access to personal information collected from and about them by domestic commercial Web sites, and maintaining adequate security for that information”<sup>7</sup>;

- The 2003 Fair and Accurate Credit Transactions Act providing consumers with annual free credit reports from the three nationwide consumer credit reporting companies;<sup>5</sup>

- California’s “Shine the light” law of 2003, according to which a business cannot disclose your personal information secretly from you to a third party for “direct marketing purposes”<sup>2</sup>; and

- A 2014 FTC report that calls for laws making data broker practices more transparent and giving consumers greater control over their personal information.<sup>9</sup>

Clearly the law favors transparency and facilitates your access to your inverse private infons.

**Market forces.** The sticky point is whether companies will share back our personal information. This information is extremely valuable to them. It gives them competitive advantages, and so it may seem implausible that companies will share it back. We contend that companies will share back personal information because *it will be in their business interests*.

Sharing back personal information can be competitively advantageous as well. Other things being equal, wouldn’t you prefer to deal with a company that shares your personal infons with you? We think so. Companies will compete on (a) how much personal data, collected and derived, is shared back and (b) how convenient that data is presented to customers.

The evolution toward sharing back personal information seems slow. This will change. Once some companies start sharing back personal data as part of their routine business, the competitive pressure will quickly force their competitors to join in. The competitors will have little choice.

There is money to be made in solving the inverse privacy problem. As sharing back personal information gains ground, the need will arise to mine large amounts of customers’ personal data

## There is money to be made in solving the inverse privacy problem.

on their behalf. The benefits of owning and processing this data will grow, especially as the data involves financial and quality-of-life domains. We anticipate the emergence of a new market for companies that compete in processing large sets of private data for the benefits of the data producers, that is, consumers.

The miners of personal data will work on behalf of consumers and compete on how helpful they are to the customers, how trustworthy they are. This emerging market will generate its own pressure on the personal data holders and potentially might find ways to benefit them as well. For example, if you shop at some retailer *R* your personal data miner *M* may show you a separate webpage devoted to *R*, suggest ways for you to save money as you shop there, and show you how *R* intends to improve your shopping experience. The last part may even be written by *R*, but—working on your behalf—*M* may also suggest to you better deals or shopping experiences elsewhere. The retailer *R* will benefit if it can beat the competition.

**Better record keeping.** Finally, technology can enhance people’s capacity to take and keep records. For example, your smartphone or wearable device may eventually become a trusted and universal recorder of many things you do. Technology will help people maintain a personal diary effortlessly.

The project “Small Data” lead by Deborah Estrin at Cornell Tech<sup>3</sup> pioneers such an approach in the domain of health. “Consider a new kind of cloud-based app that would create a picture of your health over time by continuously, securely, and privately analyzing the digital traces you generate as you work, shop, sleep, eat, exercise, and communicate.”<sup>4</sup>

The “small” in “Small Data” reflects the fact that the personal health-re-

lated data of one individual isn’t big data.<sup>18</sup> In contrast to Estrin’s work, we do not restrict attention to any particular data vertical. In our case, inversely private data of an individual tends to be on the biggish side<sup>10</sup>—recall the story of Max Schrems described earlier in this Viewpoint. □

### References

1. Acquisti, A., John, L., and Loewenstein, G. What is privacy worth? In *Privacy Papers for Policy Makers 2010*; <http://www.futureofprivacy.org/privacy-papers-2010/>
2. California S.B. 27. Shine the Light Law. Civil Code x1798.83; <http://goo.gl/zxuHUI>
3. Estrin, D. What happens when each patient becomes their own “universe” of unique medical data? *TedMed* 2013; <http://www.tedmed.com/talks/show?id=17762>
4. Estrin, D. Small data where  $n = me$ . *Commun. ACM* 57, 4 (Apr. 2014), 32–34.
5. Fair and Accurate Credit Transaction Act of 2003, Public Law 108[159, 108th Congress, 117; <http://goo.gl/PnpCkv>
6. Fair Credit Reporting Act (FCRA), 1970. Title 15 U.S. Code, sec. 1681; <http://www.ftc.gov/os/statutes/fcra.htm>
7. Federal Trade Commission. Final Report of the FTC Advisory Committee on Online Access and Security, 2000; <http://govinfo.library.unt.edu/acoas/papers/finalreport.htm>
8. Federal Trade Commission. Protecting Consumer Privacy in an Era of Rapid Change, March 2012; <http://tinyurl.com/p6r3cy4>
9. Federal Trade Commission. Data Brokers: A Call for Transparency and Accountability, May 2014; <http://alturl.com/dwwy5>
10. Gurevich Y., Haiby N., Hudis E., Wing J.M. and Ziklik E. Biggish: A solution for the inverse privacy problem. Microsoft Research MSR-TR-2016-24 (May 2016); <http://research.microsoft.com/apps/pubs/default.aspx?id=266411>
11. Gurevich, Y. and Neeman, I. Infon logic: The propositional case. *ACM Transactions on Computation Logic* 12, 2, article 9 (Jan. 2011).
12. Leon, P.G. et al. Why People Are (Un)willing to Share Information with Online Advertisers. Carnegie Mellon University, Computer Science Technical Report CMU-ISR-15-106, <http://goo.gl/RkqkhJ>
13. Orwell, G. 1984. Secker and Warburg, London, 1949.
14. Schneier, B. *Data and Goliath: The Hidden Battles to Collect Your Data and Control Your World*. Norton and Company, 2015.
15. Strom, D. Deborah Estrin wants to (literally) open source your life. *ITWorld* (May 24, 2013); <http://goo.gl/Jsd9CT>
16. U.S. Department of Health, Education, and Welfare (HEW). Records, Computers, and the Rights of Citizens. Report of the Secretary’s Advisory Committee on Automated Personal Data Systems (the HEW report), July 1973; <https://epic.org/privacy/hew1973report/>
17. Warren, S.D. and Brandeis, L.D. The Right to Privacy. *Harvard Law Review* 4, 5 (1890), 193–220.
18. Wikipedia. Big data. (Aug. 2015); <https://en.wikipedia.org/?oldid=675136834>

**Yuri Gurevich** (gurevich@microsoft.com) is Principal Researcher, Microsoft Research, and Professor Emeritus at the University of Michigan.

**Efim Hudis** (efimh@microsoft.com) is Principal Architect at Microsoft, Seattle, WA.

**Jeannette M. Wing** (wing@microsoft.com) is Corporate Vice President of Microsoft Research, Redmond, WA.

Copyright held by authors.



Watch the authors discuss their work in this exclusive *Communications* video. <http://cacm.acm.org/videos/inverse-privacy>

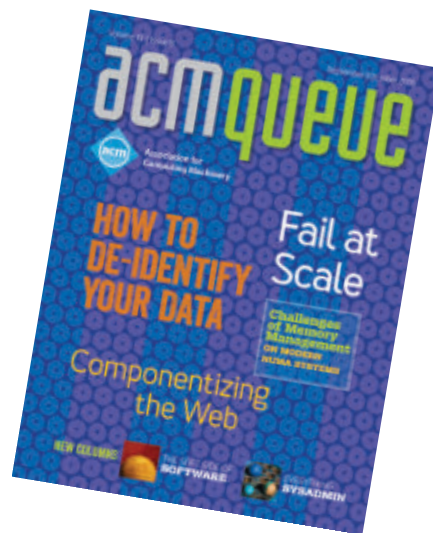
# acmqueue

## Check out the new acmqueue app

### FREE TO ACM MEMBERS

acmqueue is ACM's magazine by and for practitioners, bridging the gap between academics and practitioners of computer science. After more than a decade of providing unique perspectives on how current and emerging technologies are being applied in the field, the new acmqueue has evolved into an interactive, socially networked, electronic magazine.

Broaden your knowledge with technical articles focusing on today's problems affecting CS in practice, video interviews, roundtables, case studies, and lively columns.



## Keep up with this fast-paced world on the go. Download the mobile app.



Association for  
Computing Machinery

Desktop digital edition also available at [queue.acm.org](http://queue.acm.org).  
Bimonthly issues free to ACM Professional Members.  
Annual subscription \$19.99 for nonmembers.

---

**The accepted wisdom  
does not always hold true.**

---

**BY SACHIN DATE**

---

# Should You Upload or Ship Big Data to the Cloud?

IT IS ACCEPTED wisdom that when the data you wish to move into the cloud is at terabyte scale and beyond, you are better off shipping it to the cloud provider, rather than uploading it. This article takes an analytical look at how shipping and uploading strategies compare, the various factors on which they depend, and under what circumstances you are better off shipping rather than uploading data, and vice versa. Such an analytical determination is important to make, given the increasing availability of gigabit-speed Internet connections, along with the explosive growth in data-transfer speeds supported by newer editions of drive interfaces such as SAS and PCI Express. As this article reveals, the aforementioned “accepted wisdom” does not always hold true, and there are well-reasoned, practical recommendations

for uploading versus shipping data to the cloud.

Here are a few key insights to consider when deciding whether to upload or ship:

- ▶ A direct upload of big data to the cloud can require an unacceptable amount of time, even over Internet connections of 100Mbps (megabits per second) and faster. A convenient workaround has been to copy the data to storage tapes or hard drives and ship it to the cloud datacenter.

- ▶ With the increasing availability of affordable, optical fiber-based Internet connections, however, shipping the data via drives becomes quickly unattractive from the point of view of both cost and speed of transfer.

- ▶ Shipping big data is realistic only if you can copy the data into (and out of) the storage appliance at very high speeds and you have a high-capacity, reusable storage appliance at your disposal. In this case, the shipping strategy can easily beat even optical fiber-based data upload on speed, provided the size of data is above a certain threshold value.

- ▶ For a given value of drive-to-drive data-transfer speed, this threshold data size (beyond which shipping the data to the cloud becomes faster than uploading it) grows with every Mbps increase in the available upload speed. This growth continues up to a certain threshold upload speed. If your ISP provides an upload speed of greater or equal to this threshold speed, uploading the data will always be faster than shipping it to the cloud, no matter how big the data is.

Suppose you want to upload your video collection into the public cloud; or let's say your company wishes to migrate its data from a private datacenter to a public cloud, or move it from one datacenter to another. In a way it doesn't matter what your profile is. Given the explosion in the amount of digital information that both individuals and enterprises have to deal with, the prospect of moving big data from one place to another



over the Internet is closer than you might think.

To illustrate, let's say you have 1TB of business data to migrate to cloud storage from your self-managed datacenter. You are signed up with a business plan with your ISP that guarantees you an upload speed of 50Mbps and a download speed of 10 times as much. All you need to do is announce a short system-down-time window and begin hauling your data up to the cloud. Right?

Not quite.

For starters, you will need a whopping 47 hours to finish uploading 1TB of data at a speed of 50Mbps—and that's assuming your connection never drops or slows down.

If you upgrade to a faster—say, 100Mbps—upload plan, you can finish

the job in one day. But what if you have 2 TB of content to upload, or 4TB, or 10TB? Even at a 100Mbps sustained data-transfer rate, you will need a mind-boggling 233 hours to move 10TB of content!

As you can see, conventional wisdom breaks down at terabyte and petabyte scales. It's necessary to look at alternative, nonobvious ways of dealing with data of this magnitude.

Here are two such alternatives available today for moving big data:

- Copy the data locally to a storage appliance such as LTO (linear tape open) tape, HDD (hard-disk drive), or SSD (solid-state drive), and ship it to the cloud provider. For convenience, let's call this strategy "Ship It!"

- Perform a cloud-to-cloud transfer

of content over the Internet using APIs (application programming interfaces available) from both the source and destination cloud providers.<sup>6</sup> Let's call this strategy "Transfer It!"

This article compares these alternatives, with respect to time and cost, to the baseline technique of uploading the data to the cloud server using an Internet connection. This baseline technique is called "Upload It!" for short.

### A Real-Life Scenario

Suppose you want to upload your content into, purely for the sake of illustration, the Amazon S3 (Simple Storage Service) cloud, specifically its datacenter in Oregon.<sup>2</sup> This could well be any other cloud-storage service provided by players<sup>9</sup> in this space such as (but not limited to) Microsoft, Google, Rackspace, and IBM. Also, let's assume that your private datacenter is located in Kansas City, MO, which happens to be approximately geographically equidistant from Amazon's datacenters<sup>2</sup> located in the eastern and western U.S.

Kansas City is also one of the few places where a gigabit-speed optical-fiber service is available in the U.S. In this case, it's offered by Google Fiber.<sup>7</sup>

As of November 2015, Google Fiber offers one of the highest speeds that an ISP can provide in the U.S.: 1Gbps (gigabit per second), for both upload and download.<sup>13</sup> Short of having access to a leased Gigabit Ethernet<sup>11</sup> line, an optical fiber-based Internet service is a really, really fast way to shove bits up and down Internet pipes anywhere in the world.

Assuming an average sustained upload speed of 800Mbps on such a fiber-based connection,<sup>13</sup> (that is, 80% of its advertised theoretical maximum speed of 1Gbps), uploading 1TB of data will require almost three hours to upload from Kansas City to S3 storage in Oregon. This is actually pretty quick (assuming, of course, your connection never slows down). Moreover, as the size of the data increases, the upload time increases in the same ratio: 20TB requires 2.5 days to upload, 50TB requires almost a week to upload, and 100TB requires twice that long. At the other end of the scale, a half a petabyte of data requires two

Figure 1. Data flow when copying data to a storage appliance.

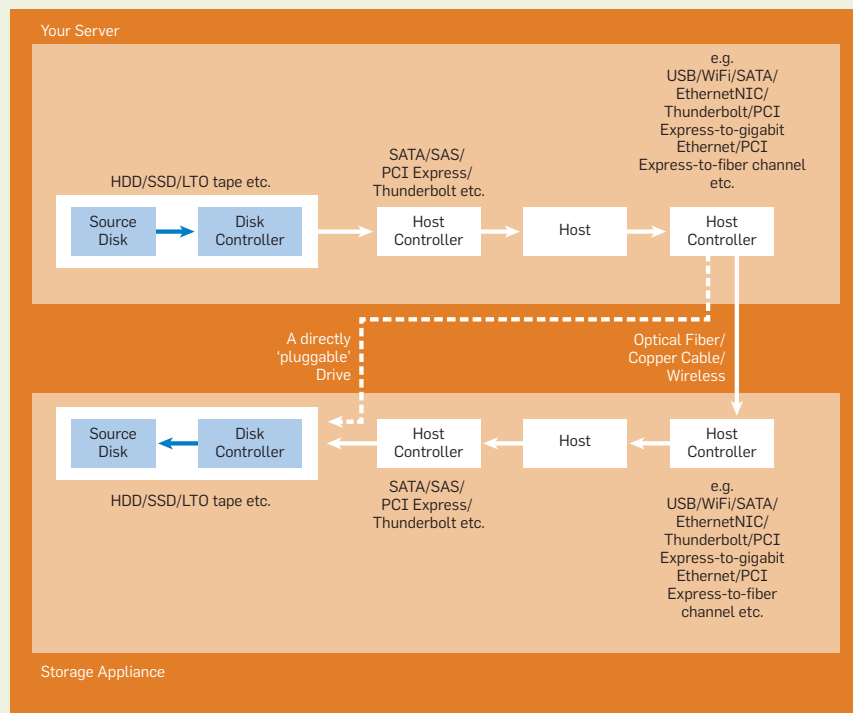


Figure 2. Data transfer speeds supported by various interfaces.

| Interface Type           | Data Transfer Speed (Gbps)                                          |
|--------------------------|---------------------------------------------------------------------|
| SATA Revision 3          | 6 <sup>17</sup>                                                     |
| SAS-3                    | 12 <sup>10</sup>                                                    |
| SuperSpeed USB (USB 3.0) | 10 <sup>20</sup>                                                    |
| PCI Express version 4    | 15.754 (single data lane), to 252.064 (16 data lanes) <sup>14</sup> |
| Thunderbolt 2            | 20 <sup>1</sup>                                                     |

months to upload. Uploading one petabyte at 800Mbps should keep you going for four months.

It's time to consider an alternative.

### Ship It!

That alternative is copying the data to a storage appliance and shipping the appliance to the datacenter, at which end the data is copied to cloud storage. This is the Ship It! strategy. Under what circumstances is this a viable alternative to uploading the data directly into the cloud?

#### The mathematics of shipping data.

When data is read out from a drive, it travels from the physical drive hardware (for example, the HDD platter) to the on-board disk controller (the electronic circuitry on the drive). From there the data travels to the host controller (a.k.a. the host bus adapter, a.k.a. the interface card) and finally to the host system (for example, the computer with which the drive is interfaced). When data is written to the drive, it follows the reverse route.

When data is copied from a server to a storage appliance (or vice versa), the data must travel through an additional physical layer, such as an Ethernet or USB connection existing between the server and the storage appliance.

Figure 1 is a simplified view of the data flow when copying data to a storage appliance. The direction of data flow shown in the figure is conceptually reversed when the data is copied out from the storage appliance to the cloud server.

Note that often the storage appliance may be nothing more than a single hard drive, in which case the data flow from the server to this drive is basically along the dotted line in the figure.

Given this data flow, a simple way to express the time needed to transfer the data to the cloud using the Ship It! strategy is shown in Equation 1, where:  $V_{\text{content}}$  is the volume of data to be transferred in megabytes (MB).

$\text{Speed}_{\text{copyIn}}$  is the *sustained* rate in MBps (megabytes per second) at which data is copied from the source drives to the storage appliance. This speed is essentially the minimum of three speeds: the speed at which the controller reads data out of the source drive and transfers it to the

**Given the explosion in the amount of digital information that both individuals and enterprises have to deal with, the prospect of moving big data from one place to another over the Internet is closer than you might think.**

host computer with which it interfaces; the speed at which the storage appliance's controller receives data from its interfaced host and writes it into the storage appliance; and the speed of data transfer between the two hosts. For example, if the two hosts are connected over a Gigabit Ethernet or a Fibre Channel connection, and the storage appliance is capable of writing data at 600MBps, but if the source drive and its controller can emit data at only 20MBps, then the effective copy-in speed can be at most 20MBps.

$\text{Speed}_{\text{copyOut}}$  is similarly the *sustained* rate in MBps at which data is copied out of the storage appliance and written into cloud storage.

$T_{\text{transit}}$  is the transit time for the shipment via the courier service from source to destination in hours.

$T_{\text{overhead}}$  is the overhead time in hours. This can include the time required to buy the storage devices (for example, tapes), set them up for data transfer, pack and create the shipment, and drop it off at the shipper's location. At the receiving end, it includes the time needed to process the shipment received from the shipper, store it temporarily, unpack it, and set it up for data transfer.

**The use of sustained data-transfer rates.** Storage devices come in a variety of types such as HDD, SSD, and LTO. Each type is available in different configurations such as a RAID (redundant array of independent disks) of HDDs or SSDs, or an HDD-SSD combination where one or more SSDs are used as a fast read-ahead cache for the HDD array. There are also many different data-transfer interfaces such as SCSI (Small Computer System Interface), SATA (Serial AT Attachment), SAS (Serial Attached SCSI), USB, PCI (Peripheral Component Interconnect) Express, Thunderbolt, and so on. Each of these interfaces supports a different theoretical maximum data-transfer speed.

Figure 2 lists the data-transfer speeds supported by a recent edition of some of these controller interfaces.

The effective copy-in/copy-out speed while copying data to/from a storage appliance depends on a number of factors:

- *Type of drive.* For example, SSDs are usually faster than HDDs partly because of the absence of any mov-

ing parts. Among HDDs, higher-RPM drives can exhibit lower seek times than lower-RPM drives. Similarly, higher areal-density (bits per surface area) drives can lead to higher data-transfer rates.

Figure 3. Growth in data transfer time, 800Mbps vs. tapes.

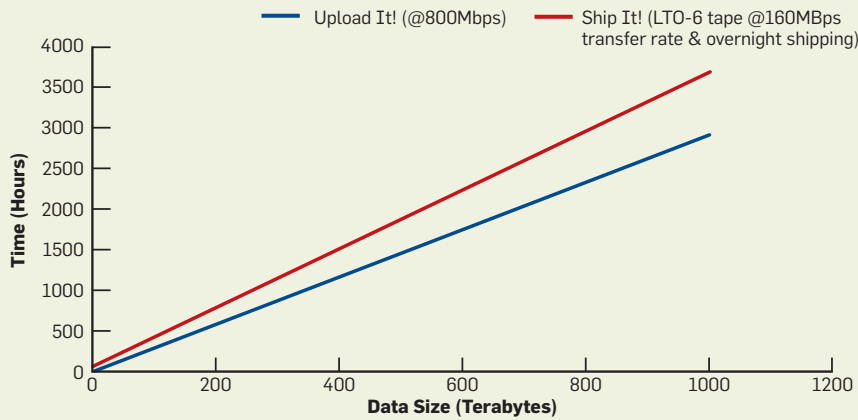


Figure 4. Growth in data transfer time, 100Mbps vs. tapes.

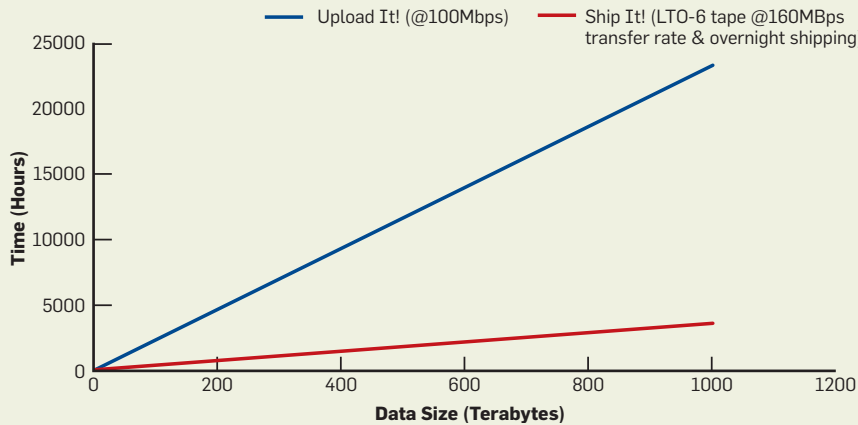
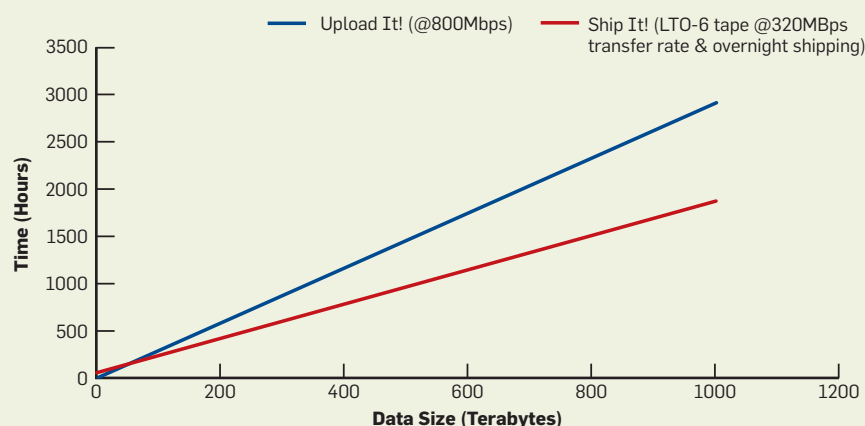


Figure 5. Growth in data transfer time, 800Mbps vs. 320MBps.



► *Configuration of the drive.* Speeds are affected by, for example, single disk versus an array of redundant disks, and the presence or absence of read-ahead caches on the drive.

► *Location of the data on the drive.*

If the drive is fragmented (particularly applicable to HDDs), it can take longer to read data from and write data to it. Similarly, on HDD platters, data located near the periphery of the platter will be read faster than data located near the spindle. This is because the linear speed of the platter near the periphery is much higher than near the spindle.

► *Type of data-transfer interface.* SAS-3 versus SATA Revision 3, for example, can make a difference in speeds.

► *Compression and encryption.* Compression and/or encryption at source and decompression and/or de-encryption at the destination reduce the effective data-transfer rate.

Because of these factors, the effective *sustained* copy-in or copy-out rate is likely to be much different (usually much less) than the burst read/write rate of either the source drive and its interface or the destination drive and its controller interface.

With these considerations in mind, let's run some numbers through Equation 1, considering the following scenario. You decide to use LTO-6 tapes for copying data. An LTO-6 cartridge can store 2.5TB of data in uncompressed form.<sup>18</sup> LTO-6 supports an uncompressed read/write data speed of 160MBps.<sup>19</sup> Let's make an important simplifying assumption that both the source drive and the destination cloud storage can match the 160MBps transfer speed of the LTO-6 tape drive (that is,  $\text{Speed}_{\text{copyIn}} = \text{Speed}_{\text{copyOut}} = 160 \text{ MBps}$ ). You choose the overnight shipping option and the shipper requires 16 hours to deliver the shipment ( $T_{\text{transit}} = 16 \text{ hours}$ ). Finally, let's factor in 48 hours of overhead time ( $T_{\text{overhead}} = 48 \text{ hours}$ ).

Plugging these values into Equation 1 and plotting the data-transfer time versus data size using the Ship It! strategy produces the maroon line in Figure 3. For the sake of comparison, the blue line shows the data-transfer time of the Upload It! strategy using a fiber-based Internet connection running at 800Mbps sustained upload rate. The figure shows

comparative growth in data-transfer time between uploading at 800Mbps versus copying it to LTO-6 tapes and shipping it overnight.

Equation 1 shows that a significant amount of time in the Ship It! strategy is spent copying data into and out of the storage appliance. The shipping time is comparatively small and constant (even if you are shipping internationally), while the drive-to-drive copy-in/copy-out time increases to a very large value as the size of the content being transferred grows. Given this fact, it's hard to beat a fiber-based connection on raw data-transfer speed, especially when the competing strategy involves copy in/copy out using an LTO-6 tape drive running at 160MBps.

Often, however, you may not be so lucky as to have access to a 1Gbps upload link. In most regions of the world, you may get no more than 100Mbps, if that much, and rarely so on a sustained basis. For example, at 100Mbps, Ship It! has a definite advantage for large data volumes, as in Figure 4, which shows comparative growth in data-transfer time between uploading at 100Mbps versus copying the data to LTO-6 tapes and shipping it overnight.

The maroon line in Figure 4 represents the transfer time of the Ship It! strategy using LTO-6 tapes, while this time the blue line represents the transfer time of the Upload It! strategy using a 100Mbps upload link. Shipping the data using LTO-6 tapes is a faster means of getting the data to the cloud than uploading it at 100Mbps for data volumes as low as four terabytes.

What if you have a much faster means of copying data in and out of the storage appliance? How would that compete with a fiber-based Internet link running at 800Mbps? With all other parameter values staying the same, and assuming a drive-to-drive copy-in/copy-out speed of 240MBps (50% faster than what LTO-6 can support), the inflection point (that is, the content size at which the Ship It! strategy becomes faster than the Upload It! strategy at 800Mbps) is around 132 terabytes. For an even faster drive-to-drive copy-in/copy-out speed of 320MBps, the inflection

point drops sharply to 59 terabytes. That means if the content size is 59TB or higher, it will be quicker just to ship the data to the cloud provider than to upload it using a fiber-based ISP running at 800Mbps.

Figure 5 shows the comparative growth in data-transfer time between uploading it at 800Mbps versus copying it at a 320MBps transfer rate and shipping it overnight.

This analysis brings up two key questions:

► If you know how much data you wish to upload, what is the minimum sustained upload speed your ISP must provide, below which you would be better off shipping the data via overnight courier?

► If your ISP has promised you a certain sustained upload speed, beyond what data size will shipping the data be a quicker way of hauling it up to the cloud than uploading it?

Equation 1 can help answer these questions by estimating how long it will take to ship your data to the datacenter. This quantity is  $(\text{Transfer Time})_{\text{hours}}$ . Now imagine uploading the same volume of data ( $V_{\text{content}}$  Megabytes), in parallel, over a network link. The question is, what is the minimum sustained upload speed needed to finish uploading everything to the datacenter in the same amount of time as shipping it there. Thus, you just have to express Equation 1's left-hand side  $(\text{Transfer Time})_{\text{hours}}$  in terms of the volume of data ( $V_{\text{content}}$  Megabytes); and the required minimum Internet connection speed ( $\text{Speed}_{\text{upload}}$  Mbps). In other words:  $(\text{Transfer Time})_{\text{hours}} = 8 \times V_{\text{content}} / \text{Speed}_{\text{upload}}$ .

Having made this substitution, let's continue with the scenario: LTO-6-based data transfer running at 160MBps, overnight shipping of 16 hours, and 48 hours of overhead time.

Figure 6. Minimum necessary upload speed for faster uploading.

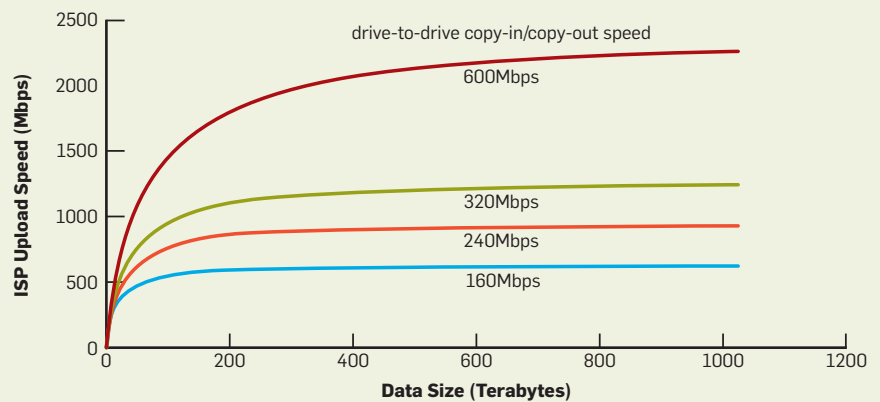
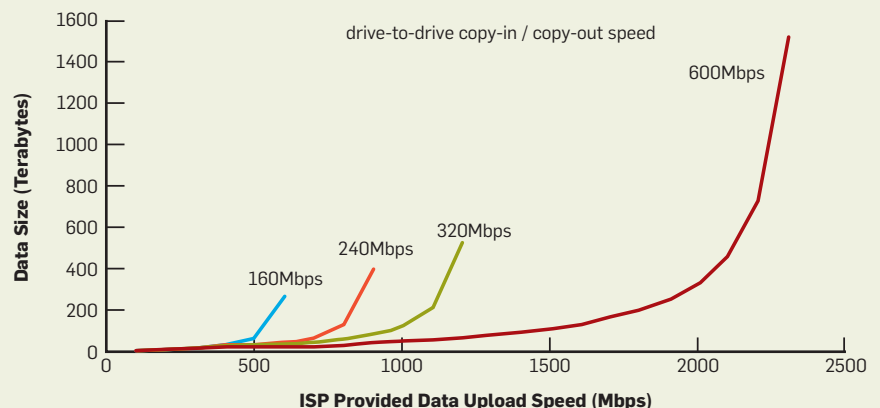


Figure 7. Maximum possible data size for faster uploading.



Also assume there is 1TB of data to be transferred to the cloud.

The aforementioned substitution reveals that unless the ISP provides a sustained upload speed ( $\text{Speed}_{\text{upload}}$ ) of at least 34.45Mbps, the data can be transferred faster using a Ship It! strategy that involves an LTO-6 tape-based data transfer running at 160MBps and a shipping and handling overhead of 64 hours.

Figure 6 shows the relationship between the volume of data to be transferred (in TB) and the minimum sustained ISP upload speed (in Mbps) needed to make uploading the data as fast as shipping it to the datacenter. For very large data sizes, the threshold ISP upload speed becomes less sensitive to the data size and more sensitive to the drive-to-drive copy-in/copy-out speeds with which it is competing.

Now let's attempt to answer the second question. This time, assume  $\text{Speed}_{\text{upload}}$  (in Mbps) is the maximum sustained upload speed that the ISP can provide. What is the maximum data size beyond which it will be quicker to ship the data to the datacenter? Once again, recall that Equation 1 helps estimate the time required ( $\text{Transfer Time}_{\text{hours}}$ ) to ship the data to the datacenter for a given data size ( $V_{\text{content}}$  MB) and drive-to-drive copy-in/copy-out speeds. If you were instead to upload  $V_{\text{content}}$  MB at  $\text{Speed}_{\text{upload}}$  Mbps over a network link, you would need  $8 \times V_{\text{content}} / \text{Speed}_{\text{upload}}$  hours. At a certain threshold value of  $V_{\text{content}}$ , these two transfer times (shipping versus upload) will become equal. Equation 1 can be rearranged to express this threshold data size as illustrated in Equation 2.

Figure 7 shows the relationship between this threshold data size and the available sustained upload speed from the ISP for different values of drive-to-drive copy-in/copy-out speeds.

Equation 2 also shows that, for a given value of drive-to-drive copy-in/copy-out speed, the upward trend in  $V_{\text{content}}$  continues up to a point where  $\text{Speed}_{\text{upload}} = 8 / \Delta T_{\text{data copy}}$ , beyond which  $V_{\text{content}}$  becomes infinite, meaning it is no longer possible to ship the data more quickly than simply uploading it to the cloud, no matter how gargantuan the data size. In this case, unless you switch to a



**Data can be transferred at various levels of granularity such as logical objects, buckets, byte blobs, files, or simply a byte stream.**



faster means of copying data in and out of the storage appliance, you are better off simply uploading it to the destination cloud.

Again, in the scenario of LTO-6 tape-based data transfer running at 160MBps transfer speed, overnight shipping of 16 hours, and 48 hours of overhead time, the upload speed beyond which it's always faster to upload than to ship your data is 640Mbps. If you have access to a faster means of drive-to-drive data copying—say, running at 320Bps—your ISP will need to offer a sustained upload speed of more than 1,280Mbps to make it speedier for you to upload the data than to copy and ship it.

### Cloud-to-Cloud Data Transfer

Another strategy is to transfer data directly from the source cloud to the destination cloud. This is usually done using APIs from the source and destination cloud providers. Data can be transferred at various levels of granularity such as logical *objects*, *buckets*, *byte blobs*, *files*, or simply a *byte stream*. You can also schedule large data transfers as batch jobs that can run unattended and alert you on completion or failure. Consider cloud-to-cloud data transfer particularly when:

- ▶ Your data is already in one such cloud-storage provider and you wish to move it to another cloud-storage provider.
- ▶ Both the source and destination cloud-storage providers offer data egress and ingress APIs.
- ▶ You wish to take advantage of the data copying and scheduling infrastructure and services already offered by the cloud providers.

Note that cloud-to-cloud transfer is conceptually the same as uploading data to the cloud in that the data moves over an Internet connection. Hence, the same speed considerations apply to it as explained previously while comparing it with the strategy of shipping data to the datacenter. Also note that the Internet connection speed from the source to destination clouds may not be the same as the upload speed provided by the ISP.

### Cost of Data Transfer

LTO-6 tapes, at 0.013 cents per GB,<sup>18</sup>

provide one of the lowest cost-to-storage ratios, compared with other options such as HDD or SSD storage. It's easy to see, however, the total cost of tape cartridges becomes prohibitive for storing terabyte and beyond content sizes. One option is to store data in a compressed form. LTO-6, for example, can store up to 6.25TB per tape<sup>18</sup> in compressed format, thereby leading to fewer tape cartridges. Compressing the data at the source and uncompressing it at the destination, however, further reduces the effective copy-in/copy-out speed of LTO tapes, or for that matter with any other storage medium. As explained earlier, a low copy-in/copy-out speed can make shipping the data less attractive than uploading it over a fiber-based ISP link.

But what if the cloud-storage provider loaned the storage appliance to you? This way, the provider can potentially afford to use higher-end options such as high-end SSDs or a combination HDD-SSD array in the storage appliance, which would otherwise be prohibitively expensive to purchase just for the purpose of transferring data. In fact, that is exactly the approach that Amazon appears to have taken with its Amazon Web Services (AWS) Snowball.<sup>3</sup> Amazon claims that up to 50TB of data can be copied from your data source into the Snowball storage appliance in less than one day. This performance characteristic translates into a sustained data-transfer rate of at least 600MBps. This kind of a data-transfer rate is possible only with very high-end SSD/HDD drive arrays with read-ahead caches operating over a fast interface such as SATA Revision 3, SAS-3, or PCI Express, and a Gigabit Ethernet link out of the storage appliance.

In fact, the performance characteristics of AWS Snowball closely resemble those of a high-performance NAS (network-attached storage) device, complete with a CPU, on-board RAM, built-in data encryption services, Gigabit Ethernet network interface, and a built-in control program—not to mention a ruggedized, tamper-proof construction. The utility of services such as Snowball comes from

the cloud provider making a very high-performance (and expensive) NAS-like device available to users to “rent” to copy-in/copy-out files to the provider's cloud. Other major cloud providers such as Google and Microsoft are not far behind in offering such capabilities. Microsoft requires you to ship SATA II/III internal HDDs for importing or exporting data into/from the Azure cloud and provides the software needed to prepare the drives for import or export.<sup>16</sup> Google, on the other hand, appears to have outsourced the data-copy service to a third-party provider.<sup>8</sup>

One final point on the cost: unless your data is in a self-managed datacenter, usually the source cloud provider will charge you for data egress,<sup>4,5,12,15</sup> whether you do a disk-based copying out of data or cloud-to-cloud data transfer. These charges are usually levied on a per-GB, per-TB, or per-request basis. There is usually no data ingress charge levied by the destination cloud provider.

## Conclusion

If you wish to move big data from one location to another over the Internet, there are a few options available—namely, uploading it directly using an ISP-provided network connection, copying it into a storage appliance and shipping the appliance to the new storage provider, and, finally, cloud-to-cloud data transfer.

Which technique you choose depends on a number of factors: the size of data to be transferred, the sustained Internet connection speed between the source and destination servers, the sustained drive-to-drive copy-in/copy-out speeds supported by the storage appliance and the source and destination drives, the monetary cost of data transfer, and to a smaller extent, the shipment cost and transit time. Some of these factors result in the emergence of threshold upload speeds and threshold data sizes that fundamentally influence which strategy you would choose. Drive-to-drive copy-in/copy-out times have enormous influence on whether it is attractive to copy and ship data, as opposed to uploading it over the Internet, especially when competing with an optical fiber-based Internet link. 

## Related articles on queue.acm.org

### How Will Astronomy Archives Survive the Data Tsunami?

G. Bruce Berriman and Steven L. Groom  
<http://queue.acm.org/detail.cfm?id=2047483>

### Condos and Clouds

Pat Helland  
<http://queue.acm.org/detail.cfm?id=2398392>

### Why Cloud Computing Will Never Be Free

Dave Durkee  
<http://queue.acm.org/detail.cfm?id=1772130>

## References

1. Apple. Thunderbolt, 2015; <http://www.apple.com/thunderbolt/>.
2. Amazon Web Services. Global infrastructure, 2015; <https://aws.amazon.com/about-aws/global-infrastructure/>.
3. Amazon. AWS Import/Export Snowball, 2015; <https://aws.amazon.com/importexport/>.
4. Amazon. Amazon S3 pricing. <https://aws.amazon.com/s3/pricing/>.
5. Google. Google cloud storage pricing; <https://cloud.google.com/storage/pricing#network-pricing>.
6. Google. Cloud storage transfer service, 2015; <https://cloud.google.com/storage/transfer/>.
7. Google. Google fiber expansion plans; <https://fiber.google.com/newcities/>.
8. Google. Offline media import/export, 2015; <https://cloud.google.com/storage/docs/offline-media-import-export>.
9. Herskowitz, N. Microsoft named a leader in Gartner's public cloud storage services for second consecutive year, 2015; <https://azure.microsoft.com/en-us/blog/microsoft-named-a-leader-in-gartners-public-cloud-storage-services-for-second-consecutive-year/>.
10. SCSI Trade Association. Serial Attached SCSI Technology Roadmap, (Oct. 14, 2015); <http://www.scsita.org/library/2015/10/serial-attached-scsi-technology-roadmap.html>.
11. IEEE. 802.3: Ethernet standards; <http://standards.ieee.org/about/get/802/802.3.html>.
12. Microsoft. Microsoft Azure data transfers pricing details; <https://azure.microsoft.com/en-us/pricing/details/data-transfers/>.
13. Ookla. America's fastest ISPs and mobile networks, 2015; <http://www.speedtest.net/awards/us/kansas-city-mo>.
14. PCI-SIG. Press release: PCI Express 4.0 evolution to 16GT/s, twice the throughput of PCI Express 3.0 technology; [http://kavi.pcisig.com/news\\_room/Press\\_Releases/November\\_29\\_2011\\_Press\\_Release\\_/](http://kavi.pcisig.com/news_room/Press_Releases/November_29_2011_Press_Release_/).
15. Rackspace. Rackspace public cloud pay-as-you-go pricing, 2015; <http://www.rackspace.com/cloud/public-pricing>.
16. Shahan, R. Microsoft Corp. Use the Microsoft Azure import/export service to transfer data to blob storage, 2015; <https://azure.microsoft.com/en-in/documentation/articles/storage-import-export-service/>.
17. The Serial ATA International Organization. SATA naming guidelines, 2015; <https://www.sata-io.org/sata-naming-guidelines>.
18. Ultrium LTO. LTO-6 capacity data sheet; [http://www.lto.org/wp-content/uploads/2014/06/ValueProp\\_Capacity.pdf](http://www.lto.org/wp-content/uploads/2014/06/ValueProp_Capacity.pdf).
19. Ultrium LTO. LTO-6 performance data sheet; [http://www.lto.org/wp-content/uploads/2014/06/ValueProp\\_Performance.pdf](http://www.lto.org/wp-content/uploads/2014/06/ValueProp_Performance.pdf).
20. USB Implementers Forum. SuperSpeed USB (USB 3.0) performance to double with new capabilities, 2013; <http://www.businesswire.com/news/home/20130106005027/en/SuperSpeed-USB-USB-3.0-Performance-Double-Capabilities>.

**Sachin Date** (<https://in.linkedin.com/in/sachindate>) looks after the Microsoft and cloud applications portfolio of e-Emphasys Technologies. Previously, he worked as a practice head for mobile technologies, an enterprise software architect, and a researcher in autonomous software agents.

Copyright held by author.  
 Publication rights licensed to ACM \$15.00.

**Reducing waste, encouraging experimentation, and making everyone happy.**

**BY THOMAS A. LIMONCELLI**

# The Small Batches Principle

**Q: WHAT DO DevOps people mean when they talk about small batches?**

A: To answer that, let's take a look at a chapter from the upcoming book *The Practice of System and Network Administration* due out later this year.

One of the themes the book explores is the *small batches principle*: it is better to do work in small batches than big leaps. Small batches permit us to deliver results faster, with higher quality and less stress.

I begin with an example that has nothing to do with system administration in order to demonstrate the general idea. Then focus on three IT-specific examples to show how the method applies and the benefits that follow.

The small batches principle is part of the DevOps methodology. It comes from the lean manufacturing movement, which is often called just-in-time (JIT) manufacturing. It can be applied to just about any kind of process, you do frequently. It also enables the MVP (minimum viable product) methodology, which involves launching a small version of a service to get early feedback that informs the decisions made later in the project.

Imagine a carpenter who needs 50 pieces of two-by-four lumber, all the same length. One could imagine cutting all 50 pieces then measuring them to verify they are all the correct size. It would be very disappointing to discover the blade shifted while making piece 10, and pieces 11 through 50 are unusable. The carpenter would have to remake 40 pieces.

A better method would be to verify the length after each piece is cut. If the blade had shifted, the carpenter would detect the problem soon after it happened, and there would be less waste.

These two approaches demonstrate big batches versus small batches. In the big-batch world the work is done in two large batches: the carpenter cut all the boards, then inspected all the boards. In the small-batch world, there are many iterations of the entire process: cut and inspect, cut and inspect, cut and inspect, and so on.

One benefit of the small-batch approach is less waste. Because an error or defect is caught immediately, the problem can be fixed before it affects other parts.

A less obvious benefit is latency. At the construction site there is a second team of carpenters who use the pieces to build a house. The pieces cannot be used until they are inspected. Using the first method, the second team cannot begin its work until all the pieces are cut and at least one piece is inspected. The chances are high the pieces will be delivered in a big batch after they have all been inspected. In the small-batch example the new pieces are delivered without this delay.



The sections that follow relate the small-batch principle to IT and show many benefits beyond reduced waste and improved latency.

### **In-House Software Deployment**

A company had a team of developers that produced a new release every six months. When a release shipped, the operations team stopped everything and deployed the release into production. The process took three or four weeks and was very stressful for all involved. Scheduling the maintenance window required complex negotiation. Testing the release was complex and required all hands on deck. The actual software installation never worked on the first try. Once deployed, a number of high-priority bugs would be discovered, and each would be fixed by various “hot patches” that follow.

Even though the deployment process was labor intensive, there was no attempt to automate it. The team had many rationalizations that justified this. The production infrastructure changed significantly between releases, thus making each release a moving target. It was believed that any automation would be useless by the next release because each release’s installation instructions were shockingly different. With each next release being so far away, there

was always a more important “burning issue” that had to be worked on first. Thus, those that did want to automate the process were told to wait until tomorrow, and tomorrow never came. Lastly, everyone secretly hoped that maybe, just maybe, the next release cycle wouldn’t be so bad. Such optimism is a triumph of hope over experience.

Each release was a stressful, painful month for all involved. Soon it was known as hell month. To make matters worse, the new software was usually late. This made it impossible for the operations team to plan ahead. In particular, it was difficult to schedule any vacation time, which made everyone more stressed and unhappy.

Feeling compassion for the team’s woes, someone proposed that releases should be done less often, perhaps every 9 or 12 months. If something is painful, it is natural to want to do it less frequently.

To everyone’s surprise the operations team suggested going in the other direction: monthly releases.

This was a big-batch situation. To improve, the company didn’t need bigger batches, it needed smaller ones.

People were shocked! Were they proposing that every month be hell month?

No, by doing it more frequently, there would be pressure to automate the pro-

cess. If something happens infrequently, there’s always an excuse to put off automating it. Also, there would be fewer changes to the infrastructure between releases. If an infrastructure change did break the release automation, it would be easier to fix the problem.

The change did not happen overnight. First the developers changed their methodology from mega releases with many new features, to small iterations, each with a few specific new features. This was a big change, and selling the idea to the team and management was a long battle.

Meanwhile, the operations team automated the testing and deployment processes. The automation could take the latest code, test it, and deploy it into the beta-test area in less than an hour. The push to production was still manual, but by reusing code for the beta rollouts it became increasingly less manual over time.

The result was the beta area was updated multiple times a day. Since it was automated, there was little reason not to. This made the process continuous, instead of periodic. Each code change triggered the full testing suite, and problems were found in minutes rather than in months.

Pushes to the production area happened monthly because they required

coordination among engineering, marketing, sales, customer support, and other groups. That said, all of these teams loved the transition from an unreliable mostly every-six-months schedule to a reliable monthly schedule. Soon these teams started initiatives to attempt weekly releases, with hopes of moving to daily releases. In the new small-batch world the following benefits were observed:

- **Features arrived faster.** While in the past a new feature took up to six months to reach production, now it could go from idea to production in days.

- **Hell month was eliminated.** After hundreds of trouble-free pushes to beta, pushing to production was easy.

- **The operations team could focus on higher-priority projects.** The team was no longer directly involved in software releases other than fixing the automation, which was rare. This freed up the team for more important projects.

- **There were fewer impediments to fixing bugs.** The first step in fixing a bug is to identify which code change was responsible. Big-batch releases had hundreds or thousands of changes to sort through to identify the guilty party. With small batches, it was usually quite obvious where to find the bug.

- **Bugs were fixed in less time.** Fixing a bug in code that was written six months ago is much more difficult than fixing a bug in code while it is still fresh in your mind. Small batches meant bugs were reported soon after the code was written, which meant developers could fix them more expertly in a shorter amount of time.

- **Developers experienced instant gratification.** Waiting six months to see the results of your efforts is demoralizing. Seeing your code help people shortly after it was written is addictive.

- **Most importantly, the operations team could finally take long vacations,** the kind that require advance planning and scheduling, thus giving them a way to reset and live healthier lives.

While these technical benefits are worthwhile, the business benefits are even more exciting:

- **Their ability to compete improved.** Confidence in the ability to add features and fix bugs led to the company becoming more aggressive about new features and fine-tuning

existing ones. Customers noticed and sales improved.

- **Fewer missed opportunities.** The sales team had been turning away business because of the company's inability to strike fast and take advantage of opportunities as they arrived. Now the company could enter markets it hadn't previously imagined.

- **Enabled a culture of automation and optimization.** Rapid releases removed common excuses not to automate. New automation brought consistency, repeatability, better error checking, and less manual labor. Plus, automation could run any time, not just when the operations team was available.

### The Failover Process

Stack Overflow's main website infrastructure is in a datacenter in New York City. If the datacenter fails or needs to be taken down for maintenance, duplicate equipment and software are running in Oregon, in stand-by mode.

The failover process is complex. Database masters need to be transitioned. Services need to be reconfigured. It takes a long time and requires skills from four different teams. Every time the process happens it fails in new and exciting ways, requiring ad-hoc solutions invented by whoever is doing the procedure.

In other words, the failover process is risky. When Tom was hired at Stack, his first thought was, "I hope I'm not on call when we have that kind of emergency."

Drunk driving is risky, so we avoid doing it. Failovers are risky, so we should avoid them, too. Right?

Wrong. There is a difference between *behavior* and *process*. Risky behaviors are inherently risky; they cannot be made less risky. Drunk driving is a risky behavior. It cannot be done safely, only avoided.

A failover is a risky process. A risky process can be made less risky by doing it more often.

The next time a failover was attempted at Stack Overflow, it took 10 hours. The infrastructure in New York had diverged from Oregon significantly. Code that was supposed to seamlessly failover had been tested only in isolation and failed when used in a real environment. Unexpected dependencies were discovered, in some cases

creating catch-22 situations that had to be resolved in the heat of the moment.

This 10-hour ordeal was the result of big batches. Because failovers happened rarely, there was an accumulation of infrastructure skew, dependencies, and stale code. There was also an accumulation of ignorance: new hires had never experienced the process; others had fallen out of practice.

To fix this problem the team decided to do more failovers. The batch size was the number of accumulated changes and other things that led to problems during a failover. Rather than let the batch size grow and grow, the team decided to keep it small. Rather than waiting for the next real disaster to exercise the failover process, they would intentionally introduce disasters.

The concept of activating the failover procedure on a system that was working perfectly may seem odd, but it is better to discover bugs and other problems in a controlled situation than during an emergency. Discovering a bug during an emergency at 4 A.M. is troublesome because those who can fix it may be unavailable—and if they are available, they are certainly unhappy to be awakened. In other words, it is better to discover a problem on Saturday at 10 A.M. when everyone is awake, available, and presumably sober.

If schoolchildren can do fire drills once a month, certainly system administrators can practice failovers a few times a year. The team began doing failover drills every two months until the process was perfected.

Each drill surfaced problems with code, documentation, and procedures. Each issue was filed as a bug and was fixed before the next drill. The next failover took five hours, then two hours, then eventually the drills could be done in an hour with zero user-visible downtime.

The process found infrastructure changes that had not been replicated in Oregon and code that did not failover properly. It identified new services that had not been engineered for smooth failover. It discovered a process that could be done only by one particular engineer. If he was on vacation or unavailable, the company would be in trouble. He was a single point of failure.

Over the course of a year all these

issues were fixed. Code was changed, better pretests were developed, and drills gave each member of the SRE (site reliability engineering) team a chance to learn the process. Eventually the overall process was simplified and easier to automate. The benefits Stack Overflow observed included:

- **Fewer surprises.** The more frequent the drills, the smoother the process became.

- **Reduced risk.** The procedure was more reliable because there were fewer hidden bugs waiting to bite.

- **Higher confidence.** The company had more confidence in the process, which meant the team could now focus on more important issues.

- **Bugs fixed faster.** The smaller accumulation of infrastructure and code changes meant each drill tested fewer changes. Bugs were easier to identify and faster to fix.

- **Bugs fixed during business hours.** Instead of having to find workarounds or implement fixes at odd hours when engineers were sleepy, they were worked on during the day when engineers were there to discuss and implement higher-quality fixes.

- **Better cross training.** Practice makes perfect. Operations team members all had a turn at doing the process in an environment where they had help readily available. No person was a single point of failure.

- **Improved process documentation and automation.** Documentation improved while the drill was running. Automation was easier to write because the repetition helped the team see what could be automated or what pieces were most worth automating.

- **New opportunities revealed.** The drills were a big source of inspiration for big-picture projects that would radically improve operations.

- **Happier developers.** There was less chance of being woken up at 4 A.M.

- **Happier operations team.** The fear of failovers was reduced, leading to less stress. More people trained in the failover procedure meant less stress on the people who had previously been single points of failure.

- **Better morale.** Employees could schedule long vacations again.

*Again, it became easier to schedule long vacations.*



**The concept of activating the failover procedure on a system that was working perfectly may seem odd, but it is better to discover bugs and other problems in a controlled situation than during an emergency.**



## The Monitoring Project

An IT department needed a monitoring system. The number of servers had grown to the point where situational awareness was no longer possible by manual means. The lack of visibility into the company's own network meant that outages were often first reported by customers, and often after the outage had been going on for hours and sometimes days.

The system administration team had a big vision for what the new monitoring system would be like. All services and networks would be monitored, the monitoring system would run on a pair of big, beefy machines, and when problems were detected a sophisticated on-call schedule would be used to determine whom to notify.

Six months into the project they had no monitoring system. The team was caught in endless debates over every design decision: monitoring strategy, how to monitor certain services, how the pager rotation would be handled, and so on. The hardware cost alone was high enough to require multiple levels of approval.

Logically the monitoring system couldn't be built until the planning was done, but sadly it looked like the planning would never end. The more the plans were discussed, the more issues were raised that needed to be discussed. The longer the planning lasted, the less likely the project would come to fruition.

Fundamentally they were having a big-batch problem. They wanted to build the perfect monitoring system in one big batch. This is unrealistic.

The team adopted a new strategy: small batches. Rather than building the perfect system, they would build a small system and evolve it.

At each step they would be able to show it to their co-workers and customers to get feedback. They could validate assumptions for real, finally putting a stop to the endless debates the requirements documents were producing. By monitoring something—anything—they would learn the reality of what worked best.

Small systems are more flexible and malleable; therefore, experiments are easier. Some experiments would work well, others would not. Because they would keep things small and flexible,

however, it would be easy to throw away the mistakes. This would enable the team to pivot, meaning they could change direction based on recent results. It is better to pivot early in the development process than to realize well into it that you have built something nobody likes.

Google calls this “launch early and often.” Launch as early as possible even if that means leaving out most of the features and launching to only a few select users. What you learn from the early launches informs the decisions later on and produces a better service in the end.

Launching early and often also gives you the opportunity to build operational infrastructure early. Some companies build a service for a year and then launch it, informing the operations team only a week prior. IT then has little time to develop operational practices such as backups, on-call playbooks, and so on. Therefore, those things are done badly. With the launch-early-and-often strategy, you gain operational experience early and you have enough time to do it right.

This is also known as the MVP strategy. As defined by Eric Ries in 2009, “The minimum viable product is that version of a new product which allows a team to collect the maximum amount of validated learning about customers with the least effort” (“Minimum Viable Product: A Guide;” <http://www.startuplessons-learned.com/2009/08/minimum-viable-product-guide.html>). In other words, rather than focusing on new functionality in each release, focus on testing an assumption in each release.

The team building the monitoring system adopted the launch-early-and-often strategy. They decided that each iteration, or small batch, would be one week long. At the end of the week they would release what was running in their beta environment to their production environment and ask for feedback from stakeholders.

For this to work they had to pick very small chunks of work. Taking a cue from Jason Punyon and Kevin Montrose (“Providence: Failure Is Always an Option;” <http://jasonpunyon.com/blog/2015/02/12/providence-failure-is-always-an-option/>), they called this “What can get done by Friday?”-driven development.



**With the launch-early-and-often strategy, you gain operational experience early and you have enough time to do it right.**



Iteration 1 had the goal of monitoring a few servers to get feedback from various stakeholders. The team installed an open-source monitoring system on a virtual machine. This was in sharp contrast to their original plan of a system that would be highly scalable. Virtual machines have less I/O and network horsepower than physical machines. Hardware could not be ordered and delivered in a one-week time frame, however. So the first iteration used virtual machines. It was what could be done by Friday.

At the end of this iteration, the team didn’t have their dream monitoring system, but they had more monitoring capability than ever before.

In this iteration they learned that SNMP (Simple Network Management Protocol) was disabled on most of the organization’s networking equipment. They would have to coordinate with the network team if they were to collect network utilization and other statistics. It was better to learn this now than to have their major deployment scuttled by making this discovery during the final big deployment. To work around this, the team decided to focus on monitoring things they did control, such as servers and services. This gave the network team time to create and implement a project to enable SNMP in a secure and tested way.

Iterations 2 and 3 proceeded well, adding more machines and testing other configuration options and features.

During iteration 4, however, the team noticed the other system administrators and managers had not been using the system much. This was worrisome. They paused to talk one-on-one with people to get some honest feedback.

What the team learned was that without the ability to have dashboards that displayed historical data, the system wasn’t very useful to its users. In all the past debates this issue had never been raised. Most confessed they had not thought it would be important until they saw the system running; others had not raised the issue because they simply assumed all monitoring systems had dashboards.

It was time to pivot.

The software package that had been the team’s second choice had very sophisticated dashboard capabilities.

More importantly, dashboards could be configured and customized by individual users. They were self-service.

After much discussion, the team decided to pivot to the other software package. In the next iteration, they set up the new software and created an equivalent set of configurations. This went very quickly because a lot of work from the previous iterations could be reused: the decisions on what and how to monitor; the work completed with the network team; and so on.

By iteration 6, the entire team was actively using the new software. Managers were setting up dashboards to display key metrics that were important to them. People were enthusiastic about the new system.

Something interesting happened around this time: a major server crashed on Saturday morning. The monitoring system alerted the sysadmin team, who were able to fix the problem before staff arrived at the office on Monday. In the past there had been similar outages but repairs had not begun until the sysadmins arrived on Monday morning, well after most employees had arrived. This showed management, in a very tangible way, the value of the system.

Iteration 7 had the goal of writing a proposal to move the monitoring system to physical machines so that it would scale better. By this time the managers who would approve such a purchase were enthusiastically using the system; many had become quite expert at creating custom dashboards. The case was made to move the system to physical hardware for better scaling and performance, and to use a duplicate set of hardware for a hot spare site in another datacenter.

The plan was approved.

In future iterations the system became more valuable to the organization as the team implemented features such as a more sophisticated on-call schedule, more monitored services, and so on. The benefits of small batches observed by the sysadmin team included:

► **Testing assumptions early prevents wasted effort.** The ability to fail early and often means the team can pivot. Problems can be fixed sooner rather than later.

► **Providing value earlier builds momentum.** People would rather have some features today than all the features tomorrow. Some monitoring is better than no monitoring. The naysayers see results and become advocates. Management has an easier time approving something that isn't hypothetical.

► **Experimentation is easier.** Often, people develop emotional attachment to code. With small batches they can be more agile because they have grown less attached to past decisions.

► **Instant gratification.** The team saw the results of their work faster, which improved morale.

► **The team was less stressed.** There is no big, scary, due date, just a constant flow of new features.

► **Big-batch debating is procrastination.** Much of the early debate had been about details and features that didn't matter or didn't get implemented.

The first few weeks were the hardest. The initial configuration required special skills. Once it was running, however, people with less technical skill or desire could add rules and make dashboards. In other words, by taking a lead and setting up the scaffolding, others can follow. This is an important point of technical leadership. Technical leadership means going first and making it easy for others to follow.

A benefit of using the MVP model is the system is always working. This is called "always being in a shippable state." The system is always working and providing benefit, even if not all the features are delivered. Therefore, if more urgent projects take the team away, the system is still usable and running. If the original big-batch plan had continued, the appearance of a more urgent project might have left the system half developed and unlaunched. The work done so far would have been for naught.

## Summary

Why are small batches better?

Small batches result in happier customers. Features get delivered with less latency. Bugs are fixed faster.

Small batches reduce risk. By testing assumptions, the prospect of future failure is reduced. More people get experience with procedures, which mean our skills improve.

Small batches reduce waste. They avoid endless debates and perfectionism that delay the team in getting started. Less time is spent implementing features that don't get used. In the event that higher-priority projects come up, the team has already delivered a usable system.

Small batches improve the ability to innovate. Because experimentation is encouraged, the team can test new ideas and keep the good ones. We can take risks. We are less attached to old pieces that must be thrown away.

Small batches improve productivity. Bugs are fixed quicker and the process of fixing them is accelerated because the code is fresher in the mind.

Small batches encourage automation. When something must happen often, excuses not to automate go away.

Small batches encourage experimentation. The team can try new things—even crazy ideas, some of which turn into competition-killing features. We fear failure less because we can easily undo a small batch if the experiment fails. More importantly, experimentation allows the team to learn something that will help them make future improvements.

Small batches make system administrators happier. We get instant gratification, and hell month disappears. It is simply a better way to work. 

## Related articles on [queue.acm.org](http://queue.acm.org)

### Breaking The Major Release Habit

Damon Poole

<http://queue.acm.org/detail.cfm?id=1165768>

### Power-Efficient Software

Eric Saxe

<http://queue.acm.org/detail.cfm?id=1698225>

### Adopting DevOps Practices in Quality Assurance

James Roche

<http://queue.acm.org/detail.cfm?id=2540984>

Thomas A. Limoncelli is a site reliability engineer at Stack Overflow Inc. in NYC. His Everything Sysadmin column appears in *acmqueue* (<http://queue.acm.org>); he blogs at [EverythingSysadmin.com](http://EverythingSysadmin.com).

Reprinted with permission. Volume 1: *The Practice of System and Network Administration*, 3rd Edition, by Thomas A. Limoncelli, Christina J. Hogan, Strata R. Chalup, Addison-Wesley Professional (Oct. 2016); <http://www.the-sysadmin-book.com/>

Copyright held by author.  
Publication rights licensed to ACM. \$15.00.

## Applying statistical techniques to operations data.

BY HEINRICH HARTMANN

# Statistics for Engineers

MODERN IT SYSTEMS collect an increasing wealth of data from network gear, operating systems, applications, and other components. This data needs to be analyzed to derive vital information about the user experience and business performance. For instance, faults need to be detected, service quality needs to be measured and resource usage of the next days and month needs to be forecasted.

*Rule #1: Spend more time working on code that analyzes the meaning of metrics, than code that collects, moves, stores and displays metrics. —Adrian Cockcroft<sup>1</sup>*

Statistics is the art of extracting information from data, and hence becomes an essential tool for operating modern IT systems. Despite a rising awareness of this fact within the community (see the quote above), resources for learning the relevant statistical methods for this domain are hard to find.

The statistics courses offered in universities usually depend on their students having prior knowledge of probability, measure, and set theory, which is a high

barrier of entry. Even worse, these courses often focus on parametric methods, such as t-tests, that are inadequate for this kind of analysis since they rely on strong assumptions on the distribution of data (for example, *normality*) that are not met by operations data.

This lack of relevance of classical, parametric statistics can be explained by history. The origins of statistics reach back to the 17<sup>th</sup> century, when computation was expensive and data was a sparse resource, leading mathematicians to spend a lot of effort to avoid calculations.

Today the stage has changed radically and allows different approaches to statistical problems. Consider this example from a textbook<sup>2</sup> used in a university statistics class:

*A fruit merchant gets a delivery of 10,000 oranges. He wants to know how many of those are rotten. To find out he takes a sample of 50 oranges and counts the number of rotten ones. Which deductions can he make about the total number of rotten oranges?*

The chapter goes on to explain various inference methods. The example translated to the IT domain could go as:

*A DB admin wants to know how many requests took longer than one second to complete. He measures the duration of all requests and counts the number of those that took longer than one second. Done.*

The abundance of computing resources has completely eliminated the need for elaborate estimations.

Therefore, this article takes a different approach to statistics. Instead of presenting textbook material on inference statistics, we will walk through four sections with *descriptive* statistical methods that are accessible and relevant to the case in point. I will discuss several visualization methods, gain a precise understanding of how to summarize data with histograms, visit classical summary statistics, and see how to replace them with robust, quantile-based alternatives. I have tried to keep prerequisite mathematical knowledge to a minimum (for example, by provid-



ing source code examples along with the formulas wherever feasible. (Disclaimer: The source code is deliberately inefficient and serves only as an illustration of the mathematical calculation. Use it at your own risk!)

## Visualizing Data

Visualization is the most essential data-analysis method. The human brain can process geometric informa-

tion much more rapidly than numbers or language. When presented with a suitable visualization, one can almost instantly capture relevant properties such as typical values and outliers.

This section runs through the basic visualization plotting methods and discusses their properties. The Python tool chain (IPython,<sup>8</sup> matplotlib,<sup>12</sup> and Seaborn<sup>17</sup>) is used here to produce the plots. The section does not dem-

onstrate how to use these tools. Many alternative plotting tools (R, MATLAB) with accompanying tutorials are available online. Source code and datasets can be found on GitHub.<sup>4</sup>

**Rug plots.** The most basic visualization method for a one-dimensional dataset  $x = [x_1, \dots, x_n]$  is the rug plot (Figure 1). It consists of a single axis on which little lines, called *rugs*, are drawn for each sample.

Rug plots are suitable for all questions where the temporal ordering of the samples is not relevant, such as common values or outliers. Problems occur if there are multiple samples with the same sample value in the dataset. Those samples will be indistinguishable in the rug plot. This problem can be addressed by adding a small random displacement (jitter) to the samples.

Despite its simple and honest character, the rug plot is not commonly used. Histograms or line plots are used instead, even if a rug plot would be more suitable.

**Histograms.** The histogram is a popular visualization method for one-

dimensional data. Instead of drawing rugs on an axis, the axis is divided into bins and bars of a certain height are drawn on top of them, so that the number of samples within a bin is proportional to the area of the bar (Figure 2).

The use of a second dimension often makes a histogram easier to comprehend than a rug plot. In particular, questions such as “Which ratio of the samples lies below  $y$ ?” can be effectively estimated by comparing areas. This convenience comes at the expense of an extra dimension used and additional choices that have to be made about value ranges and bin sizes.

Histograms are addressed in more detail later.

**Scatter plots.** The scatter plot is the most basic visualization of a two-dimensional dataset. For each pair of values  $x, y$  a point is drawn on a canvas that has coordinates  $(x, y)$  in a Cartesian coordinate system.

The scatter plot is a great tool to compare two metrics. Figure 4 plots the request rates of two different database nodes in a scatter plot. In the plot shown on top the points are mainly concentrated on a diagonal line, which means that if one node serves many requests, then the other is doing so as well. In the bottom plot the points are scattered all over the canvas, which represents a highly irregular load distribution, and might indicate a problem with the db configuration.

In addition to the fault-detection scenario outlined above, scatter plots are also an indispensable tool for capacity planning and scalability analysis.<sup>3,15</sup>

**Line plots.** The line plot is by far the most popular visualization method seen in practice. It is a special case of a scatter plot, where time stamps are plotted on the  $x$ -axis. In addition, a line is drawn between consecutive points. Figure 3 shows an example of a line plot.

The addition of the line provides the impression of a continuous transition between the individual samples. This assumption should always be challenged and taken with caution (for example, just because the CPU was idle at 1:00PM and 1:01PM, this does not mean it did not do any work in between).

Sometimes the actual data points are omitted from the visualization altogether and only the line is shown. This is a bad practice and should be avoided.

The line plot is a great tool to surface time-dependent patterns such as periods or trends. For time-independent questions—typical values, for example—other methods such as rug plots might be better suited.

**Which one to use?** Choosing a suitable visualization method depends on the question to be answered. Is time dependence important? Then a line plot is likely a good choice. If not, then rug plots or histograms are likely better tools. Do you want to compare different metrics with each other? Then consider using a scatter plot.

Figure 1. Rug plot of Web-request rates.

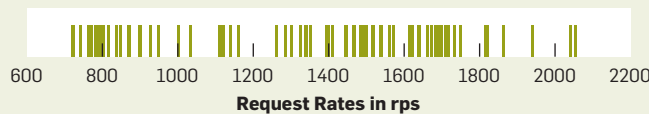


Figure 2. Histogram of Web-request rates.

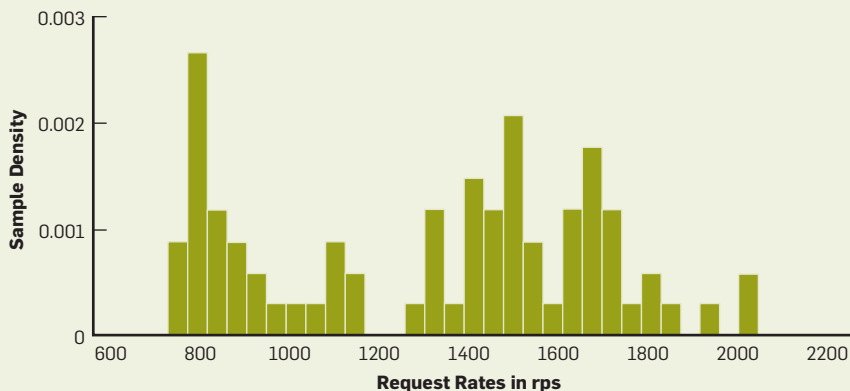
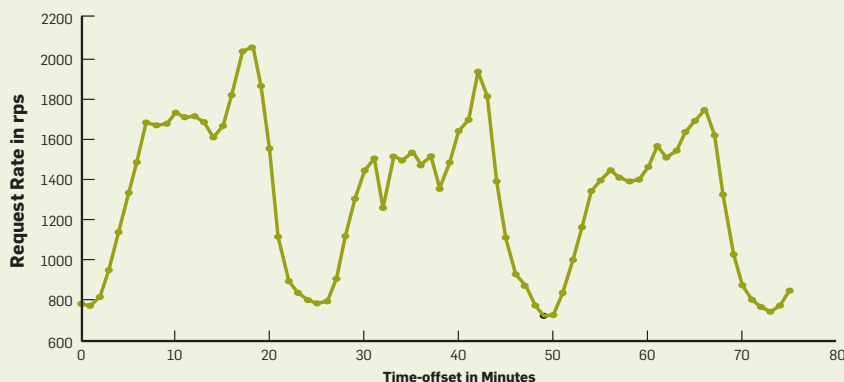


Figure 3. Line plot of Web-request rates.



Producing these plots should become routine. Your monitoring tool might be able to provide you with some of these methods already. To get the others, figure out how to export the relevant data and import it into the software tool of your choice (Python, R, or Excel). Play around with these visualizations and see how your machine data looks.

To discover more visualization methods, check out the Seaborn gallery.<sup>18</sup>

## Histograms

Histograms in IT operations have two different roles: as visualization method and as aggregation method.

To gain a complete understanding of histograms, let's start by building one for the Web request-rate data discussed previously. The listing in Figure 5 contains a complete implementation, discussed step by step here.

1. The first step in building a histogram is to choose a range of values that should be covered. To make this choice you need some prior knowledge about the dataset. Minimum and maximum values are popular choices in practice. In this example the value range is [500, 2200].

2. Next the value range is partitioned into bins. Bins are often of equal size, but there is no need to follow this convention. The bin partition is represented here by a sequence of bin boundaries (line 4).

3. Count how many samples of the given dataset are contained in each bin (lines 6–13). A value that lies on the boundary between two bins will be assigned to the higher bin.

4. Finally, produce a bar chart, where each bar is based on one bin, and the bar height is equal to the sample count divided by the bin width (lines 14–16). The division by bin width is an important normalization, since otherwise the bar area is not proportional to the sample count. Figure 5 shows the resulting histogram.

Different choices in selecting the range and bin boundaries of a histogram can affect its appearance considerably. Figure 6 shows a histogram with 100 bins for the same data. Note that it closely resembles a rug plot. On the other extreme, choosing a single bin would result in a histogram with a single bar with a height equal to the sample density.

Figure 4. Scatter plots of request rates of two database nodes.

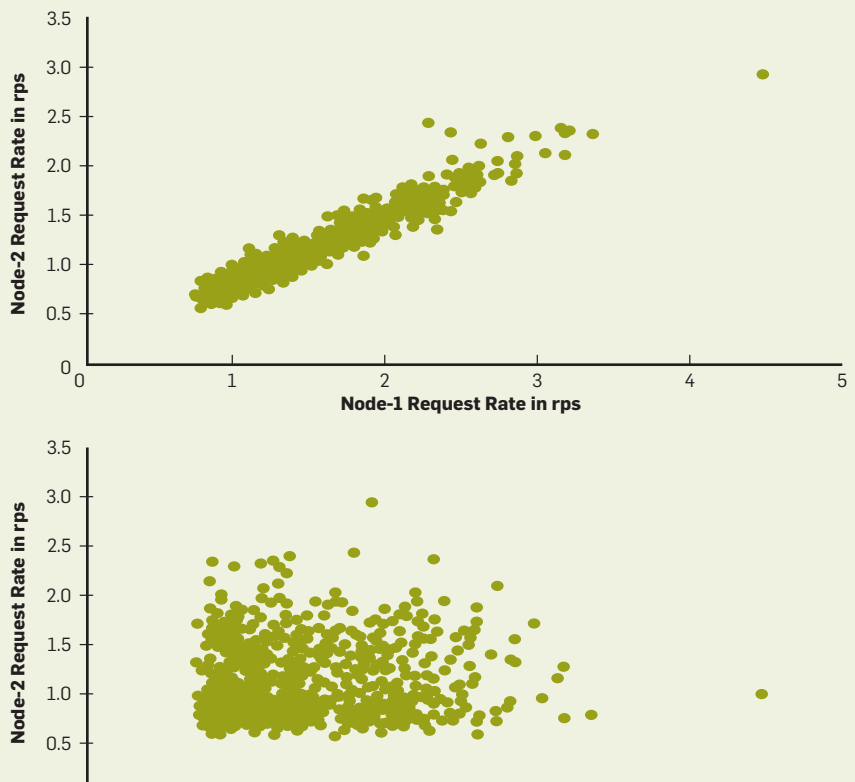


Figure 5. Result of a manual histogram implementation.

```
1 from matplotlib import pyplot as plt
2 import numpy as np
3 X = np.genfromtxt("DataSets/RequestRates.csv", delimiter=",")[:,1]
4 bins = [500, 700, 800, 900, 1000, 1500, 1800, 2000, 2200]
5 bin_count = len(bins) - 1
6 sample_counts = [0] * bin_count
7 for x in X:
8     for i in range(bin_count):
9         if (bins[i] <= x) and (x < bins[i + 1]):
10             sample_counts[i] += 1
11 bin_widths = [float(bins[i] - bins[i-1]) for i in range(1, bin_count)]
12 bin_heights = [count/width for count, width in zip(sample_counts, bin_widths)]
13 plt.bar(bins[:bin_count-1], width=bin_widths, height=bin_heights);
```



Software products make default choices for the value range and bin width. Typically the value range is taken to be the range of the data, and equally spaced bins are used. Several formulas exist for selecting the number of bins that yield “ideal” results under certain assumptions—in particular,  $n^{1/2}$  (Excel) and  $3.5\sigma/n^{1/3}$  (Scott’s rule<sup>7</sup>). In practice, these choices do not yield satisfying re-

sults when applied to operations data, such as request latencies, that contain many outliers.

#### Histogram as aggregation method.

When measuring high-frequency data such as I/O latencies, which can arrive at rates of more than 1,000 samples per second, storing all individual samples is no longer feasible. If you are willing to forget about ordering and sacrifice

some accuracy, you can save massive amounts of space by using histogram data structures.

The essential idea is, instead of storing the individual samples as a list, to use the vector of bin counts that occurs as an intermediate result in the histogram computation. The example in Figure 5 arrived at the following values:

```
sample_count = [0, 10, 8, 4, 25,
23, 4, 2]
```

The precise memory representation used for storing histograms varies. The important point is that the sample count of each bin is available.

Histograms allow approximate computation of various summary statistics, such as mean values and quantiles that will be covered next. The precision depends on the bin sizes.

Also, histograms can be aggregated easily. If request latencies are available for each node of a database cluster in histograms with the same bin choices, then you can derive the latency distribution of the whole cluster by adding the sample counts for each bin. The aggregated histogram can be used to calculate mean values and quantiles over the whole cluster. This is in contrast to the situation in which mean values or quantiles are computed for the nodes individually. It is not possible to derive, for example, the 99<sup>th</sup>-percentile of the whole cluster from the 99<sup>th</sup>-percentiles of the individual nodes.<sup>14</sup>

**High-dynamic range histogram.** An high-dynamic range (HDR) histogram provides a pragmatic choice for bin width that allows a memory-efficient representation suitable for capturing data on a very wide range that is common to machine-generated data such as I/O latencies. At the same time HDR histograms tend to produce acceptable visual representations in practice.

An HDR histogram changes the bin width dynamically over the value range. A typical HDR histogram has a bin size of 0.1 between 1 and 10, with bin boundaries: 1, 1.1, 1.2,..., 9.9, 10. Similarly, between 10 and 100 the bin size is 1, with boundaries 10, 11, 12, ..., 100. This pattern is repeated for all powers of 10, so that there are 90 bins between  $10^k$  and  $10^{k+1}$ .

The general definition is a little bit more complex and lengthy, so it is not

Figure 6. Histogram plot with value range (500, 2,200) and 100 equally sized bins.

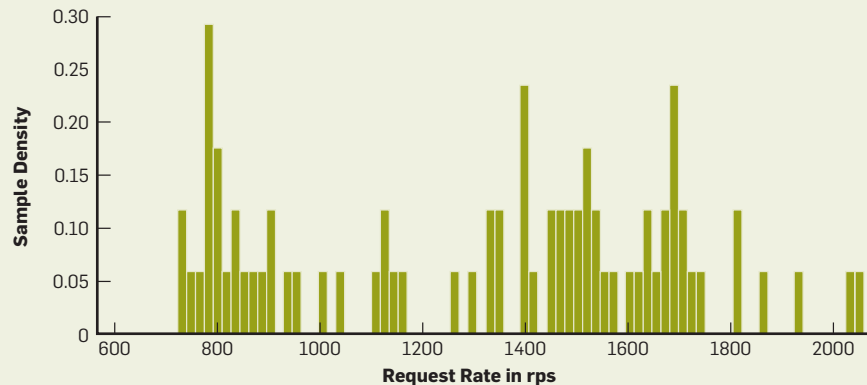


Figure 7. Request rate histogram (50 bins) presented as a heat map.

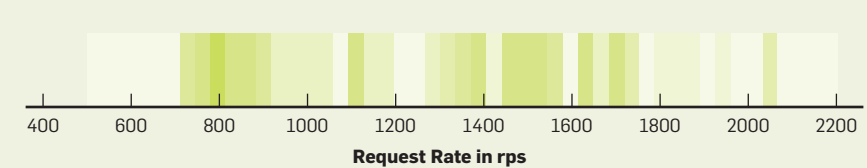


Figure 8. Request latency heat map over time in Circonus.

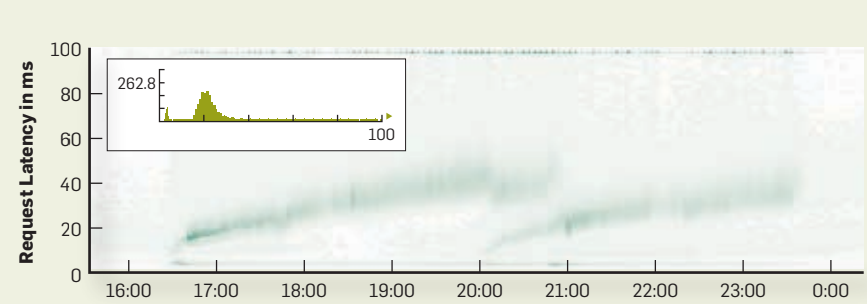
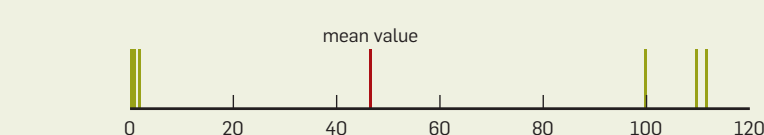


Figure 9. Rug plot of a two-modal dataset.



provided here, but interested readers can refer to <http://hdrhistogram.org/><sup>6</sup> for more details and a memory-efficient implementation.

From the previous description it should be apparent that HDR histograms span an extremely large range of values. Bin sizes are similar to float-number precisions: the larger the value, the less precision is available. In addition, bin boundaries are independent of the dataset. Hence, the aggregation technique described earlier applies to HDR histograms.

**Histograms as heat maps.** Observing the change of data distributions over time requires an additional dimension on the histogram plot. A convenient method of doing so is to represent the sample densities as a heat map instead of a bar chart. Figure 7 shows the request-rate data visualized in such a way. Light colors mean low sample density, dark colors signal high sample density.

Multiple histogram heat maps that were captured over time can be combined into a single two-dimensional heat map.

Figure 8 shows a particularly interesting example of such a visualization for a sequence of HDR histograms of Web-request latencies. Note that the distribution of the data is bimodal, with one mode constant around ~5ms and another more diffuse mode ascending from ~10ms to ~50ms. In this particular case the second mode was caused by a bug in a session handler, which kept adding new entries to a binary tree. This tree had to be traversed for each incoming request, causing extended delays. Even the logarithmic growth of the average traversal time can be spotted if you look carefully.

### Classical Summary Statistics

The aim of summary statistics is to provide a summary of the essential features of a dataset. It is the numeric equivalent of an “elevator pitch” in a business context: just the essential information without all the details.

Good summary statistics should be able to answer questions such as “What are typical values?” or “How much variation is in the data?” A desirable property is robustness against outliers. A single faulty measurement

Figure 10. Ping latency spike on a view range of 6H vs. 48H.

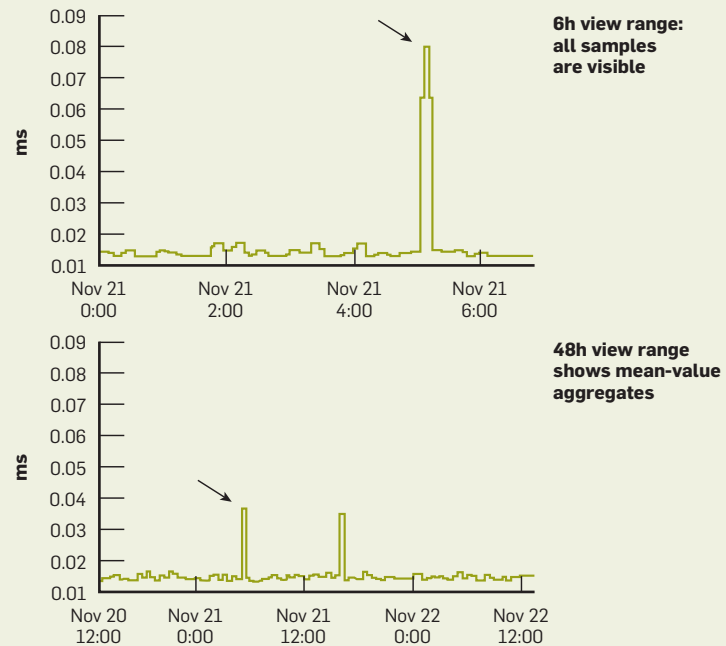
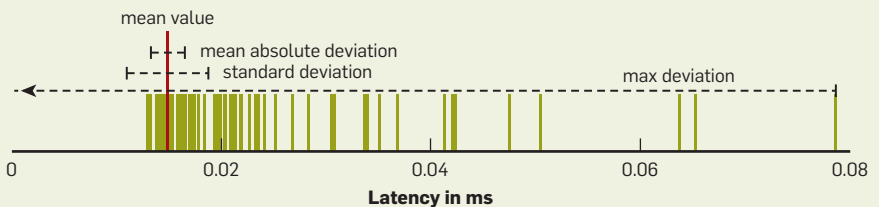


Figure 11. A request latency dataset.



should not change a rough description of the dataset.

This section looks at the classical summary statistics: mean values and standard deviations. In the next section we will meet their quantile-based counterparts, which are more robust against outliers.

**Mean value** or *average* of a dataset  $X = [x_1, \dots, x_n]$  is defined as

$$\mu(x_1, \dots, x_n) = \frac{1}{n} \sum_{i=1}^n x_i$$

or when expressed as Python code:

```
def mean(X): return sum(X) / len(X)
```

The mean value has the physical interpretation of the center of mass if weights of equal weight are placed on the points  $x_i$  on a (mass-less) axis. When the values of  $x_i$  are close together,

the mean value is a good representation of a typical sample. In contrast, when the samples are concentrated at several centers, or outliers are present, the mean value can be far from each individual data point (Figure 9).

Mean values are abundant in IT operations. One common application of mean values is data rollup. When multiple samples arrive during a sampling period of one minute, for example, the mean value is calculated as a “one-minute rollup” and stored instead of the original samples. Similarly, if data is available for every minute, but you are interested only in hour intervals, you can “roll up the data by the hour” by taking mean values.

Despite their abundance, mean values lead to a variety of problems when measuring performance of services. To quote Dogan Ugurlu from

Figure 12. The cumulative distribution function for a dataset of request rates.

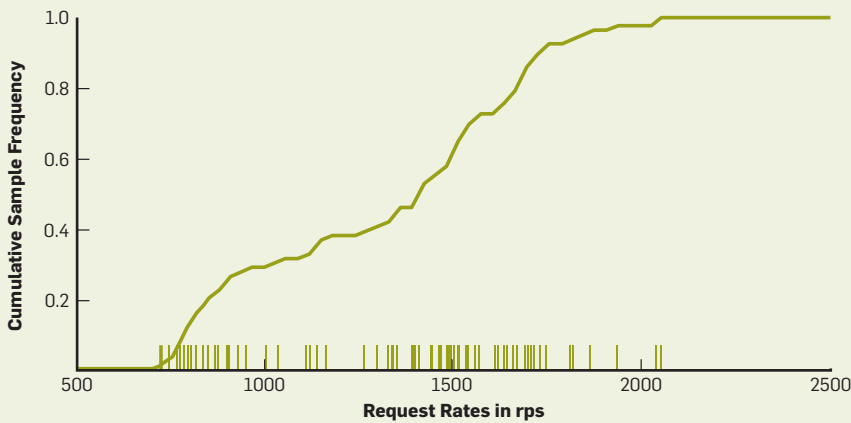


Figure 13. Histogram metric with quantile (QMIN(0.8)) over 1H windows.

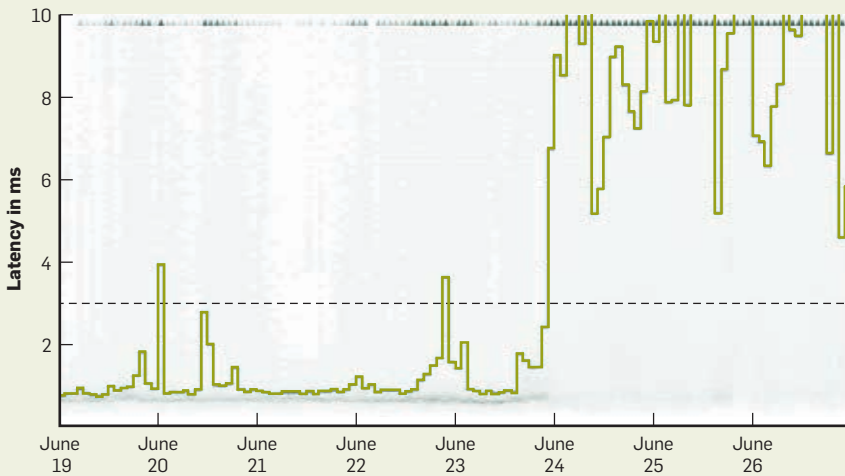
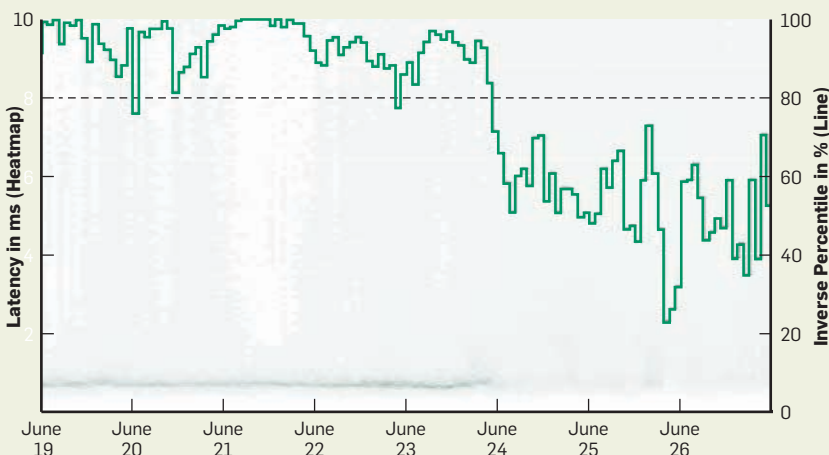


Figure 14. Histogram metric with inverse quantile CF D(3ms) over 1H windows.



Optimizely.com: “Why? Because looking at your average response time is like measuring the average temperature of a hospital. What you really care about is a patient’s temperature, and in particular, the patients who need the most help.”<sup>16</sup> In the next section we will meet median values and quantiles, which are better suited for this kind of performance analysis.

**Spike erosion.** Viewing metrics as line plots in a monitoring system often reveals a phenomenon called *spike erosion*.<sup>5</sup> To reproduce this phenomenon, select a metric (for example, ping latencies) that experiences spikes at discrete points in time and zoom in on one of those spikes and read the height of the spike at the y-axis. Now zoom out of the graph and read the height of the same spike again. Are they equal?

Figure 10 shows such a graph. The spike height has decreased from 0.8 to 0.35.

How is that possible? The result is an artifact of a rollup procedure that is commonly used when displaying graphs over long time ranges. The amount of data gathered over the period of one month (more than 40,000 minutes) is larger than the amount of pixels available for the plot. Therefore, the data has to be rolled up to larger time periods before it can be plotted. When the mean value is used for the rollups, the single spike is averaged with an increasing number of “normal” samples and hence decreases in height.

How to do better? The immediate way of addressing this problem is to choose an alternative rollup method, such as max values, but this sacrifices information about typical values. Another more elegant solution is to roll up values as histograms and display a two-dimensional heat map instead of a line plot for larger view ranges.

**Deviation measures.** Once the mean value  $\mu$  of a dataset has been established, the next natural step is to measure the deviation of the individual samples from the mean value. The following three deviation measures are often found in practice.

The *maximal deviation* is defined as  $\maxdev(x_1, \dots, x_n) = \max\{|x_i - \mu| \mid i = 1, \dots, n\}$ , and gives an upper bound for the distance to the mean in the dataset.

The *mean absolute deviation* is defined as  $mad(x_1, \dots, x_n) = \frac{1}{n} \sum_{i=1}^n |x_i - \mu|$  and is the most direct mathematical translation of a typical deviation from the mean.

The standard deviation is defined as  $stddev(x_1, \dots, x_n) = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2}$

While the intuition behind this definition is not obvious, this deviation measure is very popular for its nice mathematical properties (as being derived from a quadratic form). In fact, all three of these deviation measures fit into a continuous family of  $p$ -deviations,<sup>11</sup> which features the standard deviation in a “central” position.

Figure 11 shows the mean value and all three deviation measures for a request latency dataset. You can immediately observe the following inequalities:

$$mad(x_1, \dots, x_n) \leq stddev(x_1, \dots, x_n) \leq maxdev(x_1, \dots, x_n)$$

This relation can be shown to hold true in general.

The presence of outliers affects all three deviation measures significantly. Since operations data frequently contains outliers, the use of these deviation measures should be taken with caution or avoided all together. More robust methods (for example, interquartile ranges<sup>a</sup>) are based on quartiles, which we discuss below.

#### Caution with the standard deviation.

Many of us remember the following rule of thumb from school:

- ▶ 68% of all samples lie within one standard deviation of the mean.
- ▶ 95% of all samples lie within two standard deviations of the mean.
- ▶ 99.7% of all samples lie within three standard deviations of the mean.

These assertions rely on the crucial assumption the data is normally distributed. For operations data this is almost never the case, and the rule fails quite drastically: in the previous example more than 0.97% lie within one standard deviation of the mean value.

The following war story can be found in P.K. Janert’s book:<sup>10</sup>

*An SLA (service level agreement) for a database defined a latency outlier as a value outside of three standard deviations. The programmer who implemented the SLA check remembered the above rule*

**Despite their abundance, mean values lead to a variety of problems when measuring performance of services.**

*naively and computed the latency of the slowest 0.3 percent of the queries instead.*

*This rule has little to do with the original definition in practice. In particular, this rule labels 0.3% of each dataset blindly as outliers. Moreover, it turned out that the reported value captured long-running batch jobs that were on the order of hours. Finally, the programmer hard-coded a seemingly reasonable threshold value of ~50 seconds, and that was reported as the “three standard deviations,” regardless of the actual input.*

*The actual SLA was never changed.*

#### Quantiles and Outliers

The classical summary statistics introduced in the previous section are well suited for describing homogeneous distributions but are easily affected by outliers. Moreover, they do not contain much information about the tails of the distribution.

A quantile is a flexible tool that offers an alternative to the classical summary statistics, which is less susceptible to outliers.

Before introducing quantiles, we need to recall the following concept. The (empirical) cumulative distribution function  $CDF(y)$  for dataset  $X$ , at a value  $y$ , is the ratio of samples that are lower than the value  $y$ :

$$CDF(X, y) = \# \{i | x_i \leq y\} / \#X$$

Or expressed in Python code:

```
def CDF(X, y):
    lower_cnt = 0.0
    for x in X:
        if x <= y: lower_cnt += 1
    return lower_cnt / len(X)
```

Figure 12 shows an example for a dataset of request rates. Note  $CDF(X, y)$  takes values between 0 and 1 and is monotonically increasing as a function of  $y$ .

Now we turn to the definition of quantiles. Fix a number  $q$  between 0 and 1 and a dataset  $X$  of size  $n$ . Roughly speaking, a  $q$ -quantile is a number  $y$  that divides  $X$  into two sides, with a ratio of  $q$  samples lying below  $y$  and the remaining ratio of  $1 - q$  samples lying above  $y$ .

More formally, a  $q$ -quantile for  $X$  is a value  $y$  such that:

<sup>a</sup> [http://en.wikipedia.org/wiki/Interquartile\\_range](http://en.wikipedia.org/wiki/Interquartile_range)

- ▶ at least  $q \times n$  samples are less than or equal to  $y$ ;
- ▶ at least  $(1 - q) \times n$  samples are greater than or equal to  $y$ .

Familiar examples are the minimum, which is a 0-quantile; the maximum, which is a 1-quantile; and the median, which is a 0.5-quantile. Common names for special quantiles include percentiles for  $k/100$ -quantiles and quartiles for  $k/4$ -quantiles.

Note that quantiles are not unique. There are ways of making them unique, but those involve a choice that is not obvious. Wikipedia lists nine different choices that are found in common software products.<sup>13</sup> Therefore, if people talk about the  $q$ -quantile or the median, one should always be careful and question which choice was made.

As a simple example of how quantiles are non-unique, take a dataset with two values  $X = [10, 20]$ . Which values are medians, 0-quantiles, 0.25-quantiles? Try to figure it out yourself.

The good news is that  $q$ -quantiles always exist and are easy to compute. Indeed, let  $S$  be a sorted copy of the dataset  $X$  such that the smallest element  $X$  is equal to  $S[0]$  and the largest element of  $X$  is equal to  $S[n - 1]$ . If  $d = \text{floor}(q \times (n - 1))$ , then  $S[d]$  will have  $d+1$  samples  $S[0], \dots, S[d]$ , which are less than or equal to  $S[d]$ , and  $n - d - 1$  samples  $S[d], \dots, S[n]$ , which are greater than or equal to  $S[d]$ . It follows that  $S[d] = y$  is a  $q$ -quantile. The same argument holds true for  $d = \text{ceil}(q \times (n - 1))$ .

The following listing is a Python implementation of this construction:

```
def quantile_range(X, q):
    S = sorted(X)
    r = q * (len(X)-1)
    return (
        S[int(math.floor(r))],
        S[int(math.ceil(r))]
    )
```

It is not difficult to see this construction consists of the minimal and maximal possible  $q$ -quantiles. The notation  $Q_{\min}(X, q)$  represents the minimal  $q$ -quantile. The minimal quantile has the property  $Q_{\min}(X, q) \leq y$  if and only if at least  $n \times q$  samples of  $X$  are less than or equal to  $y$ . A similar statement holds true for the maximal

quantile when checking ratios of samples that are greater than  $y$ .

Quantiles are closely related to the cumulative distribution functions discussed in the previous section. Those concepts are inverse to each other in the following sense: If  $CDF(X, y) = q$ , then  $y$  is a  $q$ -quantile for  $X$ . Because of this property, cumulative distribution function values are also referred to as *inverse quantiles*.

### Applications to Service-Level Monitoring

Quantiles and CDFs provide a powerful method to measure service levels. To see how this works, consider the following SLA that is still commonly seen in practice: “The mean response time of the service shall not exceed three milliseconds when measured each minute over the course of one hour.”

This SLA does not do a good job of capturing the service experience of consumers. First, the requirement can be violated by a single request that takes more than 90ms to complete. Also, a long period where low overall load causes the measured request to finish within 0.1ms can compensate for a short period where lots of external requests are serviced with unacceptable response times of 100ms or more.

An SLA that captures the quality of service as experienced by the customers looks like this: 80% of all requests served by the API within one hour should complete within 3ms.

Not only is this SLA easier to formulate, but it also avoids the above problems. A single long-running request does not violate the SLA, and a busy period with long response times will violate the SLA if more than 20% of all queries are affected.

To check the SLA, here are two equivalent formulations in terms of quantiles and CFDs:

- ▶ The minimal 0.8-quantile is at most 3ms:  $Q_{\min}(X_{1h}, 0.8) \leq 3\text{ms}$ .
- ▶ The 3-ms inverse quantile is larger than 0.8:  $CDF(X_{1h}, 3\text{ms}) \geq 0.8$ .

Here  $X_{1h}$  denotes the samples that lie within a one-hour window. Both formulations can be used to monitor service levels effectively. Figure 13 shows  $Q_{\min}(X_{1h}, 0.8)$  as a line plot. Note how on June 24 the quantile rises above 3ms, indicating a violation of

the SLA. Figure 14 shows a plot of the inverse quantile  $CDF(X_{1h}, 3\text{ms})$ , which takes values on the right axis between 0% and 100%. The SLA violation manifests as the inverse quantile dropping below 80%.

Hence, quantiles and inverse quantiles give complementary views of the current service level.

### Conclusion

This article has presented an overview of some statistical techniques that find applications in IT operations. We discussed several visualization methods, their qualities and relations to each other. Histograms were shown to be an effective tool for capturing data and visualizing sample distributions. Finally, we have seen how to analyze request latencies with (inverse) percentiles. 

### References

1. Cockcroft, A. Monitorama—Please, no more Minutes, Milliseconds, Monoliths or Monitoring Tools, 2014; <http://de.slideshare.net/adriancockcroft/monitorama-please-no-more>
2. Georgii, H.-O. *Stochastic*, DeGruyter, 2002.
3. Gunther, N.J. *Guerrilla Capacity Planning*, Springer-Verlag, Berlin, 2007.
4. Hartmann, H. *Statistics for Engineers*, 2015; <https://github.com/HeinrichHartmann/Statistics-for-Engineers>.
5. Hartmann, H. *Show Me the Data*, 2016; <http://www.circonus.com/spike-erosion>.
6. HDR Histogram: A high dynamic range histogram; <http://hdrhistogram.org/>.
7. Histogram; <https://en.wikipedia.org/wiki/Histogram>.
8. IPython; <http://ipython.org>.
9. Izenman, A.J. *Modern Multivariate Statistical Techniques*, Springer-Verlag, New York, 2008.
10. Janert, P.K. *Data Analysis with Open Source Tools*, O'Reilly, 2010.
11.  $L^p$  space; [https://en.wikipedia.org/wiki/Lp\\_space](https://en.wikipedia.org/wiki/Lp_space).
12. Matplotlib; <http://matplotlib.org>.
13. Quantile; <https://en.wikipedia.org/wiki/Quantile>.
14. Schlossnagle, T. The problem with math: why your monitoring solution is wrong, 2015; <http://www.circonus.com/problem-math/>.
15. Schwarz, B. *Practical Scalability Analysis with the Universal Scalability Law*, 2015; <https://www.vividcortex.com/resources/universal-scalability-law>.
16. Ugurlu, D. The Most Misleading Measure of Response Time: Average, 2013; <https://blog.optimizely.com/2013/12/11/why-cdn-balancing/>.
17. Waskom, M. Seaborn: statistical data visualization, 2012–2015; <http://stanford.edu/~mwaskom/software/seaborn/>.
18. Waskom, M. Seaborn example gallery, 2012–2015; <http://stanford.edu/~mwaskom/software/seaborn/examples/index.html>.

**Heinrich Hartmann** is chief data scientist for the Circonus monitoring platform, leading the efforts to make Circonus the best tool to monitor APIs and services. Previously, he worked as an independent consultant for a number of different companies and research institutions.

# Are you looking for your next IT job?

## Do you need Career Advice?

The **ACM Career & Job Center** offers ACM members a host of career-enhancing benefits:

- A **highly targeted focus** on job opportunities in the computing industry
- **Access to hundreds** of industry job postings
- Resume posting **keeping you connected** to the employment market while letting you maintain full control over your confidential information
- **Job Alert system** that notifies you of new opportunities matching your criteria
- **Career coaching** and guidance available from trained experts dedicated to your success
- **Free access** to a content library of the best career articles compiled from hundreds of sources, and much more!



Visit **ACM's Career & Job Center** at:  
**<http://jobs.acm.org>**



Association for  
Computing Machinery

*Advancing Computing as a Science & Profession*

The **ACM Career & Job Center** is the perfect place to begin searching for your next employment opportunity!

Visit today at **<http://jobs.acm.org>**

**Satisfiability modulo theory solvers can help automate the search for the root cause of observable software errors.**

**BY ABHIK ROYCHOUDHURY AND SATISH CHANDRA**

# Formula-Based Software Debugging

PROGRAMMING, THOUGH A creative activity, poses strict demands on its human practitioners in terms of precision, and even talented programmers make mistakes. The effect of a mistake can manifest in several ways—as a program crash, data corruption, or unexpected output. Debugging is the task of locating the root cause of an error from its observable manifestation. It is a challenge because the manifestation of an error might become observable in a program’s execution much later than the point at which the error infected the program state in the first place. Stories abound of heroic efforts required to fix problems that cropped up unexpectedly in software that was previously considered to be working and dependable.

Given the importance of efficient debugging in overall software productivity, computer-assisted techniques

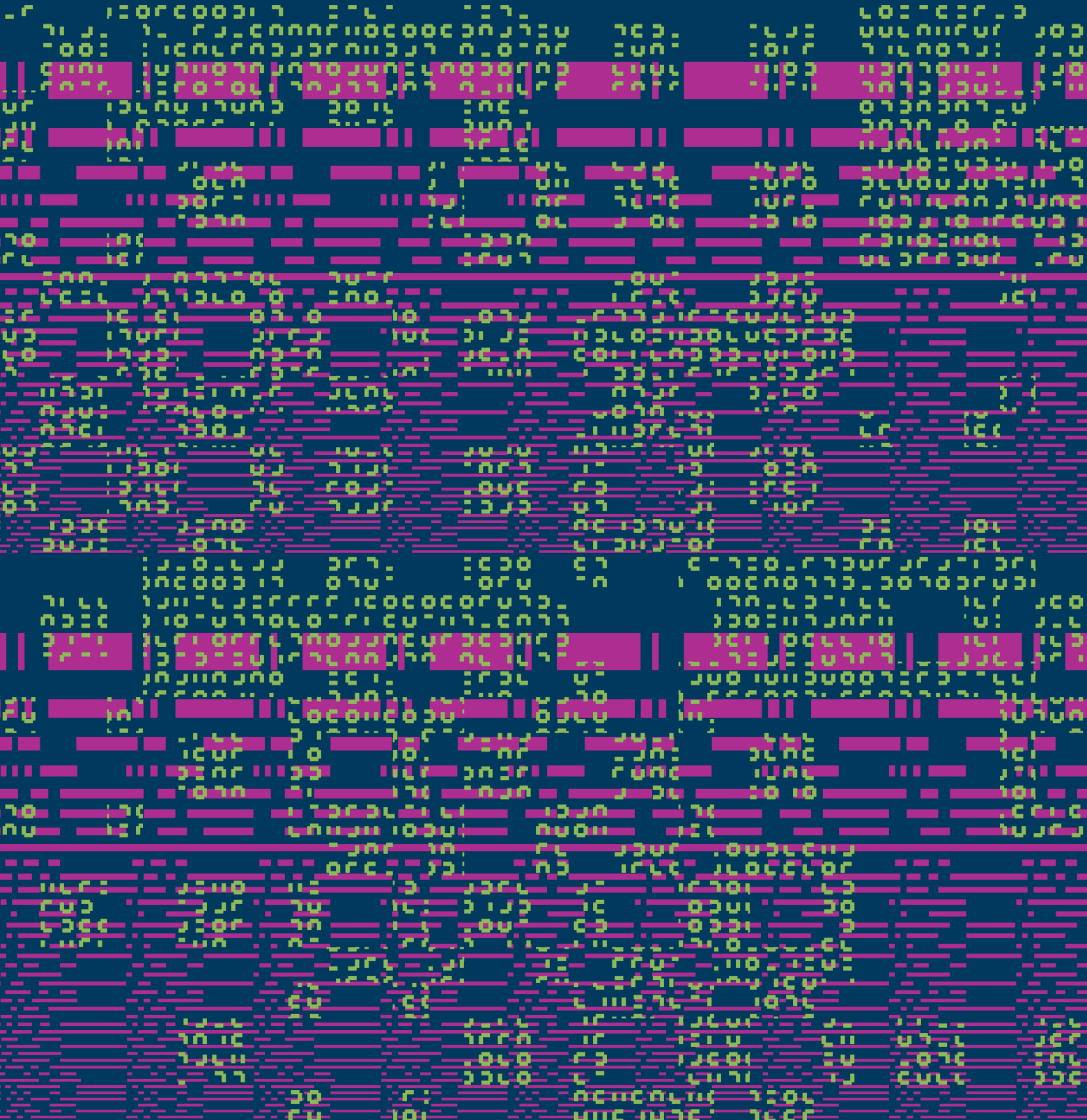
for debugging are an active topic of research. The premise of such techniques is to employ plentiful compute cycles to automatically narrow the scope of where in the source code the root cause is likely to be, thereby reducing the amount of time a programmer must spend on debugging. Such computer-assisted debugging techniques, as discussed in this article, do not promise to pinpoint the mistake in a program but only to narrow the scope of where the mistake might lurk. Such techniques are also sometimes called “fault localization” in the software-engineering literature.

We now proceed to review the major advances in computer-assisted debugging and describe one of the major challenges in debugging—lack of specifications capturing the intended behavior of the program; that is, if the intended behavior of the program is not available, how can a debugging method infer where the program went “wrong”? We next present a motivating example extracted from Address Resolution Protocol (ARP) implementation from GNU Coreutils<sup>10</sup> that serves as a running example in the article.

We then discuss at a high level how symbolic techniques can help in this direction by extracting candidate specifications. These techniques utilize some form of symbolic execution, first introduced by King.<sup>14</sup> We later describe debugging techniques that use some form of symbolic techniques to recover specifications. Table 1 outlines

## » key insights

- **The lack of explicit formal specifications of correct program behavior has long held back progress in automated debugging.**
- **To overcome this lack of formal specifications, a debugging system can use logical formula solving to attempt to find the change in a program that would make the failed test pass.**
- **Because the failure of a test in a program can also be seen as an unsatisfiable logical formula, debugging—the task of explaining failures—can thus benefit from advances in formula solving or constraint satisfaction.**



the symbolic techniques we discuss and their relation to indirect specifications. These techniques all perform symbolic analysis of various program artifacts (such as failing traces or past program versions) to help guide a programmer's search for possible causes of an observable error. Program artifacts are converted into a logical formula through symbolic analysis, and manipulation of such logical formulae

helps uncover specifications of intended program behavior. We conclude with a forward-looking view of symbolic analysis used for automated program repair, as reflected in research projects DirectFix,<sup>17</sup> Angelix,<sup>18</sup> and SemFix.<sup>20</sup>

### Computer-Assisted Debugging

There has been interest in computer-assisted debugging since at least the mid-1980s. Here, we highlight three

major ideas and refer the reader to Zeller<sup>25</sup> for a more comprehensive overview of these and other ideas in computer-assisted debugging.

The first major idea in harnessing the power of computers to aid programmers in debugging was “slicing.” Static slicing<sup>23</sup> aims to preserve only as much of the program as is necessary to retain the behavior as far as some output variable is concerned; such

controlled preservation is achieved through a static approximation of control and data dependencies. Dynamic analysis<sup>1</sup> applies the same idea but to an execution of the program on a specific input; the advantage is only that the control and data dependencies in the execution are used in the computation of the slice, leading to a more succinct and precise slice.

The second significant idea is “delta debugging”<sup>7,24</sup> in which a programmer tries to isolate the cause of a failure by systematically exploring deviations from a non-failure scenario. For example, if a new version of code breaks while the old version works, one can systematically try to isolate the specific change in the program that can be held responsible for the failure; the same idea also applies to program executions. Delta debugging takes advantage of compute cycles by systematically exploring a large number of program variations.

The third idea we highlight is “statistical fault isolation,”<sup>12,15</sup> which looks at execution profiles of passing and failing tests. If execution of a statement is strongly correlated (in a statistical sense) with only the failing tests, it is ranked highly in its suspiciousness.

Such ideas shift the burden of lo-

calizing an observable error from programmer to computer. Techniques like delta debugging rely on exploration or search over inputs or over the set of states in a trace to localize the cause of error.

Note the debugging problem implicitly contains search-based subproblems (such as the locations at which the program could be altered to avoid the observable error or which successful trace in the program you can choose to compare a given failing trace). These search problems in the debugging methods outlined earlier would be solved through various search heuristics. In contrast, the symbolic analysis-based debugging methods we present here solve these search problems by “solving logical formulae.” This new category of methods has emerged essentially out of an opportunity—the maturity and wide availability of satisfiability modulo theory (SMT) solvers.<sup>8</sup> SMT formulae are in first-order logic, where certain symbols appearing in the formula come from background theories (such as theory of integers, real numbers, lists, bitvectors, and arrays). Efficient solving of SMT formulae allows us to logically reason about

programs (static analysis) or about program artifacts like traces (dynamic analysis). The maturity of SMT solvers has made symbolic execution more practical, triggering new directions in software testing,<sup>4</sup> where symbolic execution and SMT constraint solving are used to guide the search over the huge search space of program paths for generating tests. In this article, we show how SMT solvers can be put to use in the automation of software debugging.

### Running Example

We now present a running example (see Figure 1) drawn from real-life code. It is a simplified version from a fragment of the ARP implementation in GNU Coreutils<sup>10</sup> in which a bug is introduced when a redundant assignment is added at line 5. We use it to illustrate the various debugging methods explored throughout the article.

There is an assertion in the program that captures a glimpse of the intended behavior. The rest of the intended behavior is captured through the following test cases. Without loss of generality, assume DEFAULT, NULL, ETHER, INET appearing in the program and/or test cases are predefined constants.

Test 1:

arp A INET H ETHER (passing test).  
Expected output: ETHER

Test 2:

arp A INET (failing test).  
Expected output: DEFAULT  
Actual output: NULL (assert fails)

Test 3:

arp H ETHER (passing test).  
Expected output: ETHER

The program has a redundant assignment in line 5 that changes the control flow of execution of Test 2 but not of the other tests. The violation of the intended behavior in this test is reflected in the failure of the assertion, as well as in observing an output that is different from the expected output.

The question here for a programmer is based on the failure of a test: How can the root cause be found in the failure? Being able to answer depends on a specification of the intended behavior so we can find the root cause of where the program behavior turned incorrect. On the other hand, in most ap-

**Table 1. Debugging using symbolic techniques.**

| Name                           | Symbolic Technique         | Information from       |
|--------------------------------|----------------------------|------------------------|
| BugAssist <sup>13</sup>        | Program Formula            | Internal inconsistency |
| Error Invariants <sup>10</sup> | Interpolants               | Internal inconsistency |
| Angelic Debugging <sup>5</sup> | Static Symbolic Execution  | Passing tests          |
| Darwin <sup>22</sup>           | Dynamic Symbolic Execution | Previous version       |

**Figure 1. Running example buggy program.**

```

0 hw_set = 0; hw = NULL; ap = NULL;
1 while (i = getopt(...)) {
2     switch (i) {
3         case 'A':
4             ap = getaftype(optarg);
5             hw_set = 1; // BUG: redundant line
6             break;
7         case 'H':
8             hw = gethwtype(optarg);
9             hw_set = 1;
10            break;
11    }
12 }
13 if (hw_set == 0) {
14     hw = gethwtype(DEFAULT);
15 }
16 assert(hw != NULL);
17 printf("%d", hw);

```

plication domains, programmers do not write down detailed specifications of the intended behavior of their code. Protocol specifications, even when available, are usually at a much higher level than the level of the implementation described here. In this particular case, when Test 2 fails, how does the programmer infer where the program execution turned incorrect?

The execution trace of Test 2 is as follows

```
0 hw_set = 0; hw = NULL; ap = NULL;
1 while (i = getopt(...)) {
2   switch (i) {
3     case 'A':
4       ap = getaftype(optarg);
5       hw_set = 1;
6       break;
11  } // exit switch statement
1 while (i = getopt(...)) {
12 } // exit while loop
13 if (hw_set == 0)
{ // this condition is false
16 assert(hw != NULL);
// assertion fails
```

So, when the assertion fails in line 16, which line in the program does the programmer hold responsible for this observed error? Is it line 13, where the condition checked could have been different (such as `hw_set == 1`)? If this was indeed the condition checked in line 13, Test 2 would not fail. Is it line 5, where `hw_set` is assigned? This is the line we hypothesized as the bug when we presented the buggy code to a human, but how does a computer-aided debugging method know which line is the real culprit for the observed error, and can it be fixed?

In general, for any observable error, there are several ways to fix a fault, and the definition of the fault often depends on how it is fixed. Since specifications are unavailable, have we thus reached the limit of what can be achieved in computer-assisted debugging? Fortunately, it turns out some notion of intended behavior of the program can be recovered through indirect means (such as internal inconsistency of the failed trace, passing test cases, or an older working version of the same program). In this article, we discuss debugging methods that rely on such indirect program specifications to find the root cause of an observable error.

**Fortunately,  
it turns out  
some notion of  
intended behavior  
of the program  
can be recovered  
through  
indirect means.**

## Using Satisfiability

The technique we discuss first is called “BugAssist”<sup>13</sup> in which the input and desired output information from a failing test are encoded as constraints. These input-output constraints are then conjoined with the program formula, which encodes the operational semantics of the program symbolically along all paths; see Figure 2 for an overview of conversion from a program to a formula. In the example program of Figure 1, we produce the following formula ( $\phi$ ):

$$\begin{aligned} \phi = & \text{arg}[1] = A \wedge \text{arg}[2] = \text{INET} \wedge \text{arg}[3] = \text{NULL} \\ & \wedge \text{hw\_set}_0 = 0 \wedge \text{hw}_0 = \text{NULL} \wedge \text{ap}_0 = \text{NULL} \\ & \wedge i_1 = \text{arg}[1] \wedge i_1 \neq \text{NULL} \\ & \wedge \text{guard}_3 = (i_1 = A) \\ & \wedge \text{ap}_4 = \text{arg}[2] \\ & \wedge \text{hw\_set}_5 = 1 \\ & \wedge \text{ap}_{11} = \text{guard}_3 ? \text{ap}_4 : \text{ap}_0 \\ & \wedge \text{hw\_set}_{11} = \text{guard}_3 ? \text{hw\_set}_5 : \text{hw\_set}_0 \\ & \wedge i'_1 = \text{arg}[3] \wedge i'_1 = \text{NULL} \\ & \wedge \text{guard}_{13} = (\text{hw\_set}_{11} = 0) \\ & \wedge \text{hw}_{14} = \text{DEFAULT} \\ & \wedge \text{hw}_{15} = \text{guard}_{13} ? \text{hw}_{14} : \text{hw}_0 \\ & \wedge \text{hw}_{15} \neq \text{NULL} \wedge \text{hw}_{15} = \text{DEFAULT} \end{aligned}$$

The *arg* elements refer to the input values, similar to *argv* of the C language; here, the inputs are for Test 2. The last line of clauses represents the expectation of Test 2. The remainder of the formula represents the program logic. (For brevity, we have omitted the parts of the formula corresponding to the case ‘H’, as it does not matter for this input.) The variable  $i'_1$  refers to the second time the loop condition while is evaluated, at which point the loop exits. We use  $=$  to indicate assignment and  $==$  to indicate equality test in the program, though for the satisfiability of the formula both have the same meaning.

The formula  $\phi$ , though lengthy, has one-to-one correspondence to the trace of Test 2 outlined earlier. Since the test input used corresponds to a failing test, the formula is unsatisfiable.

The BugAssist tool tries to infer what went wrong by trying to make a large part of the formula satisfiable, accomplishing it through MAX-SAT<sup>19</sup> or MAX-SMT solvers. As the name suggests, a MAX-SAT solver returns the largest possible satisfiable sub-formula of a formula; the sub-formula omits

**Figure 2. Conversion of program to formula; the formula encodes, using guards, possible final valuations of variables.**

```

1 input y; // initially x = z = 0
2 if (y > 0) {
3   z = y * 2;
4   x = y - 2;
5   x = x - 2; }
6 if (z == x)
7   output("How did I get here");
8 else if (z > x)
9   output("Error");

```

Here is the corresponding formula

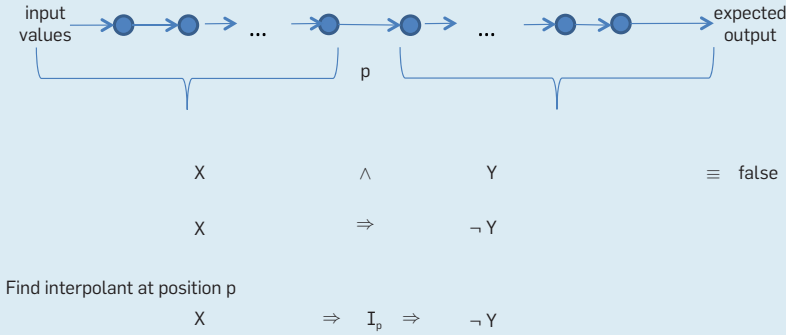
```

guard2 = (y > 0)
 $\wedge z_3 = y * 2$ 
 $\wedge x_4 = y - 2$ 
 $\wedge x_5 = x_4 - 2$ 
 $\wedge z_6 = \text{guard}_2 ? z_3 : 0$ 
 $\wedge x_6 = \text{guard}_2 ? x_5 : 0$ 
 $\wedge \text{guard}_6 = (z_6 == x_6)$ 
 $\wedge \text{guard}_8 = (z_6 > x_6)$ 
 $\wedge \text{output} = \text{guard}_6 ? \text{How} \dots : (\text{guard}_8 ? \text{Error} : \text{nil})$ 

```

In it, variables are given a subscript based on the line on which an instance is assigned. Guard variables denote conditions that regulate values of variables when potentially different values of a variable reach a branch point. For example, *guard<sub>2</sub>* regulates the value of *z<sub>6</sub>* based on whether the default initial value or the assignment to *z<sub>3</sub>* reaches it.

**Figure 3. Using interpolants to analyze error traces.**



certain conjuncts of the original formula. Moreover, the MAX-SAT solver is instructed that certain constraints are *hard* constraints, in that they cannot be omitted. Constraints on input and output are typically hard. The solvers will find the maximum part of the formula, which is satisfiable, thereby suggesting the minimal portion of the program that needs to be changed to make the formula satisfiable. In this sense, the technique attempts to use internal inconsistency to help a programmer, as an indirect source of specification.

We now illustrate how BugAssist would work on the aforementioned formula. We first mark the clauses related to *args*, as well as the final constraints on *hw<sub>15</sub>* as hard. The MAX-SAT solver could first highlight the constraint *guard<sub>13</sub>* =

(*hw<sub>set</sub><sub>11</sub>* == 0) as the one to be omitted for making the rest of the formula satisfiable. The reader can verify that setting the (now) unbound variable *guard<sub>13</sub>* to true will make *hw<sub>15</sub>* equal to *hw<sub>14</sub>*, satisfying the output constraints. In terms of the program, this corresponds to an attempt to fix the program at line 13

```

13: if (hw_set == 1) { // FIX: changed guard

```

Even though this fix passes Test 2, it will fail previously passing tests like Test 1, thereby introducing regressions. The fix is thus incorrect. However, BugAssist does not vet any of the code highlighted by the technique, relying instead on the programmer to assess whether the suggested con-

straint corresponds to a good fix. Realizing this potential regression, the programmer using BugAssist would mark *guard<sub>13</sub>* = (*hw<sub>set</sub><sub>11</sub>* == 0) as a hard constraint. Marking as a hard constraint indicates to the solver it should explore other lines in the program as possible causes of error.

The MAX-SAT solver will then highlight another (set of) constraints, say, *hw<sub>set</sub><sub>5</sub>* = 1, meaning just this constraint can be omitted (or changed) to make the overall formula satisfiable. The reader can verify this forces *guard<sub>13</sub>* to be true. This corresponds to identifying line 5 in Figure 1 as the place to be fixed. Here is the correct fix

```

5: // hw_set = 1; FIX: deleted line

```

Interestingly, from the perspective of the satisfiability of the formula, changing the value assigned to *hw\_set* from 1 to 0 is also a plausible but not robust fix, meaning it can fail other tests, as we show later in the article.

The BugAssist technique tries to extract the reason for failure through analysis of the error trace. Extraction is done iteratively, by successively finding minimal portions of the formula, the omission or alteration of which can make the error trace formula satisfiable. In some sense, the complement of MAX-SAT reported by BugAssist in repeated iterations form legitimate explanations of the observed failure in the error trace being examined. As may be observed even from our simple example, the technique may report several potential faults. It is thus not so much as a one-shot fault-localization method as it is an “iterative exploration” of the potential locations where a change could avert the error in question. The iterative exploration is guided by the computation of the maximum satisfiable portion of an unsatisfiable formula that captures the program failure.

### Using Interpolants

An alternative method, called “error invariants,”<sup>9</sup> tries to find the reason for failure by examining error propagation in the different positions of the error trace. Identifying error root-cause is achieved by computing interpolant formula at each position of the error trace. The notion of interpolant<sup>16</sup> requires some explanation. Given a logical implication  $X \Rightarrow Y$  involving first-order

logic formulae  $X$  and  $Y$ , an interpolant is a formula  $I$  satisfying

$$X \Rightarrow I \Rightarrow Y$$

The formula  $I$  is expressed through the common vocabulary of  $X$  and  $Y$ . Figure 3 outlines the use of interpolants for analyzing error traces. The error trace formula is a logical conjunction of the input valuation, the effect of each program statement on the trace, and the expectation about output.

Given any position  $p$  in the trace, if we denote the formula from locations prior to  $p$  as  $X$ , and the formula from locations in the trace after  $p$  as  $Y$ , clearly  $X \wedge Y$  is false. Thus  $\neg(X \wedge Y)$  holds, meaning  $\neg X \vee \neg Y$  holds, meaning  $X \Rightarrow \neg Y$  holds. The interpolant  $I_p$  at position  $p$  in the error trace will thus satisfy  $X \Rightarrow I_p \Rightarrow \neg Y$ . Such an interpolant can be computed at every position  $p$  in the error trace for understanding the reason behind the observable error.

Let us now understand the role of interpolants in software debugging. We first work out our simple example, then conceptualize use of the logical formula captured by interpolant in explaining software errors. In our running example, the interpolants at the different positions are listed in Table 2, including interpolant formula after each statement. Note there are many choices of interpolants at any position in the trace, and we have shown the weakest interpolant in this simple example. The trace here again corresponds to the failing execution of Test 2 on the program in Figure 1, and we used the same statements earlier in this article on BugAssist.

What role do interpolants play in explaining an error trace? To answer, we examine the sequence of interpolants computed for the error trace in our simple example program, looking at the second column in Table 2 and considering only non-repeated formulae:

$arg[1] = A$   
 $arg[1] = A \wedge hw_0 = NULL$   
 $i_1 = A \wedge hw_0 = NULL$   
 $guard_3 = true \wedge hw_0 = NULL$   
 $guard_3 = true \wedge hw_0 = NULL \wedge hw_{set_5} = 1$   
 $hw_0 = NULL \wedge hw_{set_{11}} = 1$   
 $hw_0 = NULL \wedge guard_{13} = false$   
 $hw_{15} = NULL$

The sequence of interpolants here

shows the propagation of the error via the sequence of variables  $arg[1]$ ,  $hw_0$ ,  $i_1$ , and so on. Propagation through both data and control dependence is tracked. Propagation through data dependence corresponds to an incorrect value being passed from one variable to another through assignment statements. Propagation through control dependence corresponds to an incorrect valuation of the *guard* variables, leading to an incorrect set of statements being executed. Both types of propagation are captured in the interpolant sequence computed over the failed trace. The interpolant at a position  $p$  in the error trace captures the “cause” for failure expressed in terms of variables that are live at  $p$ . Computing the interpolant at all locations of the error trace allows the developer to observe the error-propagation chain.

The other important observation to make is program statements that do not alter the interpolant are irrelevant to the explanation of the error. In Table 2, statements marked with a • are not relevant to explaining the failure of Test 2. For example, anything relevant only to the computation of  $ap$  is ignored, an approach similar to backward dynamic slicing, though an interpolation-based technique is more general than dynamic slicing. The remaining statements form a minimal error explanation. Once again, the technique uses

the internal inconsistency of the faulty execution to figure out possible causes of error. Note, choosing interpolants must be done with care—as interpolants in general are not unique—for the method to be effective in filtering away irrelevant statements.<sup>6,9</sup> Furthermore, the scalability of these methods is a concern today due to the huge length of real-life traces and the slowness of interpolating provers.<sup>16</sup>

### Using Passing Tests

Another technique, called “Angelic Debugging”<sup>5</sup> first proposed in 2011, explores the relationship between fault localization and fix localization rather closely, following the philosophy of defining a possible fault in terms of how it is fixed. In it, we explore the set of potential repairs that will make an observable error disappear. Since the landscape of syntactic repairs to try out is so vast, the technique finds, via symbolic execution and constraint solving, a value that makes the failing tests pass while continuing to pass the passing tests. Crucially, the technique utilizes the information contained in the passing tests to help identify fix locations. The technique proceeds in two steps. In the first, it attempts to find all the expressions in the program that are candidates for a fix; that is, a change made in that expression can possibly fix the program. The second step rules out those fix loca-

**Table 2. Interpolant computation at each statement; the statements marked with • have the property that they do not alter the interpolant.**

| Statement                                           | Interpolant after statement                               |
|-----------------------------------------------------|-----------------------------------------------------------|
| $arg[1] = A$                                        | $arg[1] = A$                                              |
| $arg[2] = INET$ •                                   | $arg[1] = A$                                              |
| $arg[3] = NULL$ •                                   | $arg[1] = A$                                              |
| $hw_{set_0} = 0$ •                                  | $arg[1] = A$                                              |
| $hw_0 = NULL$                                       | $arg[1] = A \wedge hw_0 = NULL$                           |
| $ap_0 = NULL$ •                                     | $arg[1] = A \wedge hw_0 = NULL$                           |
| $i_1 = arg[1]$                                      | $i_1 = A \wedge hw_0 = NULL$                              |
| $guard_3 = (i_1 == A)$                              | $guard_3 = true \wedge hw_0 = NULL$                       |
| $ap_4 = arg[2]$ •                                   | $guard_3 = true \wedge hw_0 = NULL$                       |
| $hw_{set_5} = 1$                                    | $guard_3 = true \wedge hw_0 = NULL \wedge hw_{set_5} = 1$ |
| $ap_{11} = guard_3 ? ap_4 : ap_0$ •                 | $guard_3 = true \wedge hw_0 = NULL \wedge hw_{set_5} = 1$ |
| $hw_{set_{11}} = guard_3 ? hw_{set_5} : hw_{set_0}$ | $hw_0 = NULL \wedge hw_{set_{11}} = 1$                    |
| $i'_1 = arg[3]$ •                                   | $hw_0 = NULL \wedge hw_{set_{11}} = 1$                    |
| $guard_{13} = (hw_{set_{11}} == 0)$                 | $hw_0 = NULL \wedge guard_{13} = false$                   |
| $hw_{14} = DEFAULT$ •                               | $hw_0 = NULL \wedge guard_{13} = false$                   |
| $hw_{15} = guard_{13} ? hw_{14} : hw_0$             | $hw_{15} = NULL$                                          |
| $hw_{15} \neq NULL$                                 | false                                                     |

tions for which changing the expression would make previously passing tests fail. It does so *without* knowing any proposed candidate fix, again because the landscape of syntactic fixes is so vast; rather, it works on just the basis of a candidate fix “location.”

Consider again the failing execution of Test 2 on the program in Figure 1. We illustrate how the technique works, focusing first on statement 5. The technique conceptually replaces the right-hand-side expression by a “hole” denoted by  $!!$ , meaning an as-yet-unknown expression

```
5: hw_set = !!
```

The interpretation of  $!!$  is that an angel would supply a suitable value for it if it is possible to make the execution go to completion, hence the name of the technique. The angel is simulated by a symbolic execution tree (see Figure 4 for details on how to compute a symbolic execution tree), looking for a path along which the path formula is satisfiable. In our running example, the symbolic execution comes up with the following environment when going through the true branch at line 13 and expecting a successful termination

$$e_{13T} = [hw_0 = NULL, i = A, ap = INET, hw\_set = \alpha, guard_{13} = true, hw_{14} = DEFAULT, hw_{15} = DEFAULT; \alpha = 0 \wedge hw_{15} \neq NULL \wedge hw_{15} = DEFAULT]$$

and the following when going through the false branch

$$e_{13F} = [hw_0 = NULL, i = A, ap = INET, hw\_set = \alpha, guard_{13} = false, hw_{15} = NULL; \alpha \neq 0 \wedge hw_{15} \neq NULL \wedge hw_{15} = DEFAULT]$$

$\alpha$  represents the angelic value assigned at line 5. Recall we carried out concrete execution up to statement 5, with the same input as shown in formula  $\phi$  earlier.

$e_{13T}$  has a satisfiable condition when  $\alpha$  is 0, whereas  $e_{13F}$  is not satisfiable due to the conflict on the value of  $hw_{15}$ . The execution can thus be correct in case the *guard* at line 13 evaluates to true.

Focusing next on statement 13, we find making the condition itself angelic,

## The scalability of symbolic analysis-based debugging methods crucially depends on the scalability of SMT constraint solving.

we can find a plausible successful execution of the program

```
13: if (!!)
```

We omit the formulae, but they are similar to ones shown earlier in this article.

We now show the second step of the method. It rules out those fix locations for which changing the expression would make previously passing tests fail. Given a candidate fix location, it asks the following question for each of the passing inputs: Considering the proposed fix location a hole ( $!!$ ), is there a way for the angel to provide a value for the hole that is *different* from the value observed at that location in normal execution on that input? If it is possible, then the fix location is a plausible fix location. The technique provides the angelic values by using symbolic execution.

First we consider the fix location of line 5, right-hand side. For Test 1, the hole in the fix location (line 5 of Figure 1) will be replaced by  $\alpha$ , and  $\alpha \neq 1$  added to the constraint, to represent difference from the value observed here in normal execution. More formally, the symbolic environment at line 5 will be

$$e_5 = [hw_0 = NULL, i = A, ap = INET, hw\_set = \alpha; \alpha \neq 1]$$

From here on, symbolic execution will find a path that succeeds. Likewise, for Test 2 and Test 3. The passing tests thus accept the proposed fix location as a plausible one.

Now consider the fix location of line 13, where we want the branch to have a different outcome. For Test 1, the environment at line 13 will be

$$e_{13} = [hw_8 = ETHER, hw\_set = 1, \dots, guard_{13} = \alpha; \alpha \neq false]$$

There is no successful execution given this environment. Test 1 therefore rules out line 13 as a plausible fix location, deeming no syntactic variation of the condition is likely to fix the program.

Although the technique determines plausible fix locations and not fixes themselves, going from a fix location to a fix is not straightforward. Consider a candidate syntactic fix a human could provide for line 5. For example, using the

following fix at line 5 works for tests 1–3

```
5: hw_set = 0
```

The astute reader will notice this particular fix is not an ideal fix. Given another test

```
Test 4:  arp H ETHER A INET
        Expected output: ETHER
        Actual output: DEFAULT
```

This test will fail with the proposed fix, even though the location of the fix is the correct one. The correct fix is to eliminate the effect of line 5 altogether

```
5: hw_set = hw_set; // or delete the
statement
```

The example reflects the limitations in attempting to fix a program when working with an incomplete notion of “specification.”

### Using Other Implementations

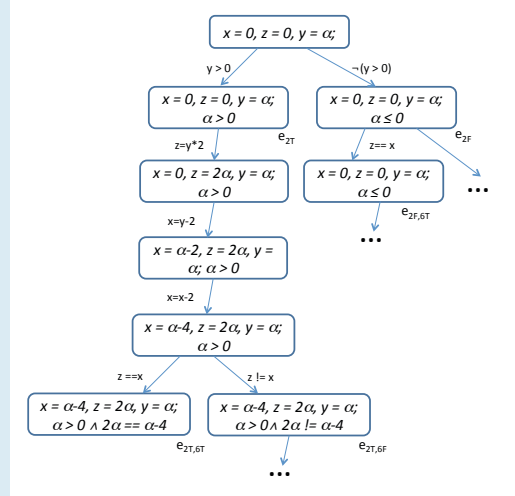
Programs are usually not written from scratch; rather, versions of a program are gradually checked in. When we introduce changes into a previously working version (where we say the version is “working,” since all test cases pass), certain passing tests may thus fail. To debug the cause of the observed failure in such a failed test  $t$ , we can treat the behavior of  $t$  in the previous working version as a description of the programmer’s intention.

The technique presented in Qi et al.,<sup>22</sup> called “Darwin,” developed in 2009, executes the failing test  $t$  in the previous program version  $P$ , as well as the current program version  $P'$ . It then calculates the path conditions of  $t$  in both program versions along the execution trace of  $t$  in both program versions; see Figure 5 for explanation of how such path conditions are computed. Calculating path conditions leads to path conditions  $f$  and  $f'$ . One can then solve the formula  $f \wedge \neg f'$  to find test input  $t'$  that follows the same path as  $t$  in the previous program version and a different path in the current program version. The execution of  $t'$  in the current program version  $P'$  can then be compared with the execution of the failing test  $t$  in current version  $P'$  in terms of differences in their control flow. That is, the behavior of  $t'$  in cur-

**Figure 4. Symbolic execution tree.**

Consider again the code fragment in Figure 2. Suppose input  $y$  is a symbolic input, with an unknown value of, say,  $\alpha$ . In symbolic execution,<sup>14</sup> the store maps variables that may be concrete values or symbolic expressions. At an assignment, the store is updated with the evaluation of right-hand-side expression, which may be a symbolic expression. At a branch, if the decision involves a symbolic expression, both sides of the branch are executed in separate “threads,” with corresponding branch conditions taken into account.

At line 2, two threads of symbolic execution will be created. In the first, the environment  $e_{2T}$  is  $[x = 0, z = 0, y = \alpha; \alpha > 0]$ , and the other,  $e_{2F}$ , is  $[x = 0, z = 0, y = \alpha; \alpha \leq 0]$ ; note we included the conditions encountered on the path thus far in the environment. These conditions appear following the semicolon (see also Figure 5.). Here is the symbolic execution tree for Figure 2



At line 6,  $e_{2T}$  forks into:  $e_{2T,6T}$  and  $e_{2T,6F}$ .  $e_{2T,6T}$  will be  $[x = \alpha - 4, z = \alpha * 2, y = \alpha; \alpha > 0 \wedge \alpha - 4 = \alpha * 2]$ , which will be discarded since the condition is unsatisfiable.

Symbolic execution tree construction is similar to the program formula construction in Figure 2. For this reason, it is also called “static symbolic execution.” The difference is, in program formula, threads were merged with control-flow join points, whereas in symbolic execution tree, there is no merging.

**Figure 5. Illustration of path conditions.**

Consider yet again the program in Figure 2. Suppose we want to find the path condition of the only way to reach the error statement, or the path  $\langle 1, 2, 3, 4, 5, 6, 8, 9 \rangle$ . We traverse forward along the sequence of statements in the given path, starting with a null formula and gradually build it up. All variables start with symbolic values. At any point during the traversal of the trace, we maintain an assignment store and a logical formula. The result is

- For every assignment, we update the symbolic assignment store; and
- For every branch, we conjoin the branch condition—or its converse if the branch is not taken—with the path condition; while doing so, we use the symbolic assignment store for every variable appearing in the branch condition.

At the end of the path, the logical formula captures the path condition. For the example path  $\langle 1, 2, 3, 4, 5, 6, 8, 9 \rangle$  in the given program, the path condition can be calculated as shown in the table here. Whenever the input satisfies  $y > 0 \wedge 2y \neq y - 4 \wedge 2y > y - 4$ , the program execution will trace exactly this path.

|   | Assignment store          | Logical Formula                                |
|---|---------------------------|------------------------------------------------|
| 1 | $\{\}$                    | $true$                                         |
| 2 | $\{\}$                    | $y > 0$                                        |
| 3 | $\{z, 2y\}$               | $y > 0$                                        |
| 4 | $\{z, 2y\}, (x, y-2)\}$   | $y > 0$                                        |
| 5 | $\{z, 2y\}, (x, y-2-2)\}$ | $y > 0$                                        |
| 6 | $\{z, 2y\}, (x, y-2-2)\}$ | $y > 0 \wedge 2y \neq y - 4$                   |
| 8 | $\{z, 2y\}, (x, y-2-2)\}$ | $y > 0 \wedge 2y \neq y - 4 \wedge 2y > y - 4$ |
| 9 | $\{z, 2y\}, (x, y-2-2)\}$ | $y > 0 \wedge 2y \neq y - 4 \wedge 2y > y - 4$ |

Note this form of symbolic execution is similar to the one in Figure 4; the difference is one path is already given here, so there is no execution tree to be explored. This is sometimes also called “dynamic symbolic execution.”

rent program  $P'$  is taken as the “specification” against which the behavior of the failing test  $t$  is compared for difference in control flow.

Such methods are based on semantic analysis, rather than a completely syntactic analysis of differences across program versions (such as running a `diff` between program versions). Being based on semantic analysis these debugging methods can analyze two versions with substantially different implementations and locate causes of error.

To illustrate this point, consider the fixed Address Resolution Protocol (ARP) implementation—Figure 1 with line 5 deleted—we discussed earlier as the reference version. This program will pass the test Test 2. Now assume a buggy program implementation with a substantially different programming style but with intention to accomplish the same ARP (see Figure 6). The test Test 2 fails in this implementation

Test 2:

```
arp A INET (failing test).
Expected output: DEFAULT
Observed output: INET
```

First of all, a simple `diff` of the program versions cannot help since almost the entire program will appear in the `diff`. A careful comparison of the two implementations shows the logic of the protocol the programmer would want to implement has been mangled in this implementation. The computation of `get_hwtype(DEFAULT)` has been (correctly) moved. However, the compu-

tation of `get_hwtype(optarg)`<sup>a</sup> is executed under an incorrect condition, leading to the failure in the test execution. A redundant check `ap != NULL` has slipped into line 8.

We now step through the localization of the error. For the test `arp A INET` the path condition in the reference version is (since  $i$  is set to `arg[1]`) as follows

$$f \equiv \text{arg}[1] = A$$

The path condition in the buggy implementation is as follows (since  $i$  is set to `arg[1]` and `ap` is set to `arg[2]` (via `optarg`)

$$f' \equiv \text{arg}[1] = A \wedge (\text{arg}[2] \neq \text{NULL} \vee \text{arg}[1] = H)$$

The negation of  $f'$  is the following disjunction

$$\neg f' \equiv \text{arg}[1] \neq A \vee \neg(\text{arg}[2] \neq \text{NULL} \vee \text{arg}[1] = H)$$

$f \wedge \neg f'$  thus has two possibilities to consider, one for each disjunct in  $\neg f'$

1.  $\text{arg}[1] = A \wedge \text{arg}[1] \neq A$
2.  $\text{arg}[1] = A \wedge \neg(\text{arg}[2] \neq \text{NULL} \vee \text{arg}[1] = H)$

The first formula is not satisfiable, and the second one simplifies to

$$\text{arg}[1] = A \wedge \text{arg}[2] = \text{NULL} \wedge \text{arg}[1] \neq H$$

A satisfying assignment to this second formula is an input that shows the essential control-flow path deviation—in the defective program—from the failure-inducing input. This formula

<sup>a</sup> Assume `get_hwtype(INET)` returns `INET`.

is satisfiable because `arg[2] = NULL` is satisfiable, pointing to the condition being misplaced in the code. This is indeed where the bug lurks. Even though the entire coding style and the control flow in the buggy implementation was quite different from the reference implementation, the debugging method is thus able to ignore the differences in coding style in the buggy implementation. Note `arg[2] = NULL` is contributed by the branch condition `ap != NULL`, a correlation an automated debugging method can keep track of. The method thus naturally zooms into the misplaced check `ap != NULL` through a computation of satisfiability of the deviations of the failing test’s path condition.

One may question the choice of considering the past version as a specification of what the program is supposed to achieve, a question that arises because the software requirements of an earlier program version may differ from the requirements of the current program version. However, the comparison works as long as the program features are common to both versions and the requirements are unchanged.

## Perspectives

We have explored symbolic execution and constraint solving for software debugging. We conclude by discussing scalability and applicability of the presented techniques, along with possible future research directions.

**Scalability and applicability.** The scalability of the techniques described here need greater investigation. Due to the semantic nature of the analysis, symbolic execution-based debugging of regression errors<sup>22</sup> has been adapted to multiple settings of regression, resulting in wide applicability, including regressions in a program version as opposed to previous version; regression in an embedded software (such as Linux Busybox) as opposed to reference software (such as GNU Coreutils);<sup>3</sup> and regression in a protocol implementation (such as miniweb Web server implementing the http protocol) as opposed to a reference implementation of the protocol, as in the Apache Web server. In these large-scale applications, the symbolic analysis-based regres-

**Figure 6.** Assume line 5 in Figure 1 was removed yielding the correct program for the code fragment; here we show a different implementation of the same code.

```
1 hw_set = 0; hw = NULL;
2 while (i = getopt(...)) // this sets optarg
3 {
4     if (i == 'A') {
5         ap = getaftype(optarg);
6         hw = get_hwtype(DEFAULT);
7     }
8     if (ap != NULL || i == 'H') {
9         hw = get_hwtype(optarg);
10        hw_set = 1;
11    }
12 }
13 assert(hw != NULL);
14 print hw;
...
```

sion debugging methods of Banerjee et al.<sup>3</sup> and Qi et al.<sup>22</sup> localized the error to within 10 lines of code in fewer than 10 minutes.

Among the techniques covered here, the regression-debugging methods<sup>3,22</sup> have shown the greatest scalability, with the other techniques being employed on small-to-moderate-scale programs. Moreover, the scalability of symbolic analysis-based debugging methods crucially depends on the scalability of SMT constraint solving.<sup>8</sup> Compared to statistical fault-localization techniques, which are easily implemented, symbolic execution-based debugging methods still involve more implementation effort, as well as greater execution time overheads. While we see much promise due to the growth in SMT solver technology, as partly evidenced by the scalability of the regression-debugging methods, more research is needed in symbolic analysis and SMT constraint solving to enhance the scalability and applicability of these methods.

Note for all of the presented debugging methods, professional programmers need user studies to measure programmer productivity gain that might be realized through these methods. Parnin and Orso<sup>21</sup> highlighted the importance of user studies in evaluating debugging methods. The need for user studies may be even more acute for methods like BugAssist that provide an iterative exploration of the possible error causes, instead of providing a final set of diagnostic information capturing the lines likely to be the causes of errors. Finally, for the interpolant-based debugging method, the issue of choosing suitable interpolants needs further study, a topic being investigated today (such as by Albarghouthi and McMillan<sup>2</sup>).

**Other directions.** Related to our topic of using symbolic execution for software debugging, we wish to say symbolic execution can also be useful for “bug reproduction,” as shown by Jin and Orso.<sup>11</sup> The bug-reproduction problem is different from both test generation and test explanation. Here, some hints may be reported about the failing execution by on-the-field users in the form of a crash report, and these

hints can be combined through symbolic execution to construct a failing execution trace.

Finally, the software-engineering community has shown much interest in building semiautomated methods for program repair. It would be interesting to see how symbolic execution-based debugging methods can help develop program-repair techniques. The research community is already witnessing development of novel program-repair methods based on symbolic execution and program synthesis.<sup>17,18,20</sup>

### Acknowledgments

We would like to acknowledge many discussions with our co-authors in Banerjee et al.,<sup>3</sup> Chandra et al.,<sup>5</sup> and Qi et al.<sup>22</sup> We also acknowledge discussions with researchers in a Dagstuhl seminar on fault localization (February 2013) and a Dagstuhl seminar on symbolic execution and constraint solving (October 2014). We further acknowledge a grant from the National Research Foundation, Prime Minister's Office, Singapore, under its National Cybersecurity R&D Program (Award No. NRF2014NCR-NCR001-21) and administered by the National Cybersecurity R&D Directorate. 

### References

1. Agrawal, H. and Horgan, J. Dynamic program slicing. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation* (White Plains, NY, June 20–22). ACM Press, New York, 1990.
2. Albarghouthi, A. and McMillan, K.L. Beautiful interpolants. In *Proceedings of the 25th International Conference on Computer-Aided Verification, Lecture Notes in Computer Science 8044* (Saint Petersburg, Russia, July 13–19). Springer, 2013.
3. Banerjee, A., Roychoudhury, A., Harlie, J.A., and Liang, Z. Golden implementation-driven software debugging. In *Proceedings of the 18th International Symposium on Foundations of Software Engineering* (Santa Fe, NM, Nov. 7–11). ACM Press, New York, 2010, 177–186.
4. Cadar, C. and Sen, K. Symbolic execution for software testing: Three decades later. *Commun. ACM* 56, 1 (Jan. 2013), 82–90.
5. Chandra, S., Torlak, E., Barman, S., and Bodik, R. Angelic debugging. In *Proceedings of the 33rd International Conference on Software Engineering* (Honolulu, HI, May 21–28). ACM Press, New York, 2011, 121–130.
6. Christ, J., Ermis, E., Schaff, M., and Wies, T. Flow sensitive fault localization. In *Proceedings of the 14th International Conference on Verification Model Checking and Abstract Interpretation* (Rome, Italy, Jan. 20–22). Springer, 2013.
7. Cleve, H. and Zeller, A. Locating causes of program failures. In *Proceedings of the 27th International Conference on Software Engineering* (St. Louis, MO, May 15–21). ACM Press, New York, 2005.
8. de Moura, L. and Björner, N. Satisfiability modulo theories: Introduction and applications. *Commun. ACM* 54, 9 (Sept. 2011), 69–77.
9. Ermis, E., Schaff, M., and Wies, T. Error invariants. In *Proceedings of the 18th International Symposium on Formal Methods, Lecture Notes in Computer Science* (Paris, France, Aug. 27–31). Springer, 2012.
10. GNU Core Utilities; <http://www.gnu.org/software/coreutils/coreutils.html>
11. Jin, W. and Orso, A. BugRedux: Reproducing field failures for in-house debugging. In *Proceedings of the 34th International Conference on Software Engineering* (Zürich, Switzerland, June 2–9). IEEE, 2012.
12. Jones, J.A., Harrold, M.J., and Stasko, J.T. Visualization of test information to assist fault localization. In *Proceedings of the 24th International Conference on Software Engineering* (Orlando, FL, May 19–25). ACM Press, New York, 2002.
13. Jose, M. and Majumdar, R. Cause clue clauses: Error localization using maximum satisfiability. In *Proceedings of the 32nd International Conference on Programming Language Design and Implementation* (San Jose, CA, June 4–8). ACM Press, New York, 2011, 437–446.
14. King, J.C. Symbolic execution and program testing. *Commun. ACM* 19, 7 (July 1976) 385–394.
15. Liblit, B., Naik, M., Zheng, A.X., Aiken, A., and Jordan, M.I. Scalable statistical bug isolation. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation* (Chicago, IL, June 12–15). ACM Press, New York, 2005, 15–26.
16. McMillan, K.L. An interpolating theorem prover. *Theoretical Computer Science* 345, 1 (Nov. 2005), 101–121.
17. Mehtaev, S., Yi, J., and Roychoudhury, A. DirectFix: Looking for simple program repairs. In *Proceedings of the 37th IEEE/ACM International Conference on Software Engineering* (Firenze, Italy, May 16–24). IEEE, 2015, 448–458.
18. Mehtaev, S., Yi, J., and Roychoudhury, A. Angelix: Scalable multiline program patch synthesis via symbolic analysis. In *Proceedings of the 38th International Conference on Software Engineering* (Austin, TX, May 14–22). ACM Press, New York, 2016.
19. Morgado, A., Heras, F., Liffiton, M., Planes, J., and Marques-Silva, J. Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints* 18, 4 (2013), 478–534.
20. Nguyen, H.D.T., Qi, D., Roychoudhury, A., and Chandra, S. SemFix: Program repair via semantic analysis. In *Proceedings of the 35th International Conference on Software Engineering* (San Francisco, CA, May 18–26). IEEE/ACM, 2013, 772–781.
21. Parnin, C. and Orso, A. Are automated debugging techniques actually helping programmers? In *Proceedings of the 20th International Symposium on Software Testing and Analysis* (Toronto, ON, Canada, July 17–21). ACM Press, New York, 2011, 199–209.
22. Qi, D., Roychoudhury, A., Liang, Z., and Vaswani, K. DARWIN: An approach for debugging evolving programs. *ACM Transactions on Software Engineering and Methodology* 21, 3 (2012).
23. Weiser, M. Program slicing. *IEEE Transactions on Software Engineering* 10, 4 (1984), 352–357.
24. Zeller, A. Yesterday my program worked. Today it fails. Why? In *Proceedings of the Seventh Joint Meeting of European Software Engineering Conference and ACM SIGSOFT Symposium on Foundations of Software Engineering, Lecture Notes in Computer Science* (Toulouse, France, Sept. 1999). Springer, 1999, 253–267.
25. Zeller, A. *Why Programs Fail: A Guide to Systematic Debugging*. Elsevier, 2006.

**Abhik Roychoudhury** (abhik@comp.nus.edu.sg) is a professor of computer science in the School of Computing at the National University of Singapore and an ACM Distinguished Speaker and leads the TSUNAMI center, a software-security research effort funded by the Singapore National Research Foundation.

**Satish Chandra** (schandra@acm.org) leads the advanced programming tools research team at Samsung Research America, Mountain View, CA, and is an ACM Distinguished Scientist.

DOI:10.1145/2854146

**Google's monolithic repository provides a common source of truth for tens of thousands of developers around the world.**

BY RACHEL POTVIN AND JOSH LEVENBERG

# Why Google Stores Billions of Lines of Code in a Single Repository

EARLY GOOGLE EMPLOYEES decided to work with a shared codebase managed through a centralized source control system. This approach has served Google well for more than 16 years, and today the vast majority of Google's software assets continues to be stored in a single, shared repository. Meanwhile, the number of Google software developers has steadily increased, and the size of the Google codebase has grown exponentially (see Figure 1). As a result, the technology used to host the codebase has also evolved significantly.

This article outlines the scale of that codebase and details Google's custom-built monolithic source repository and the reasons the model was chosen. Google uses a homegrown version-control system to host one large codebase visible to, and used by, most of the software developers in the company. This centralized system is the foundation of many of Google's developer workflows. Here, we provide background on the systems and workflows that make feasible managing and working productively with such a large repository. We explain Google's "trunk-based development" strategy and the support systems that structure workflow and keep Google's codebase healthy, including software for static analysis, code clean-up, and streamlined code review.

## Google-Scale

Google's monolithic software repository, which is used by 95% of its software developers worldwide, meets the definition of an ultra-large-scale<sup>a</sup> system, providing evidence the single-source repository model can be scaled successfully.

The Google codebase includes approximately one billion files and has a history of approximately 35 million commits spanning Google's entire 18-year existence. The repository contains 86TB<sup>a</sup> of data, including approximately

<sup>a</sup> Total size of uncompressed content, excluding release branches.

## » key insights

- Google has shown the monolithic model of source code management can scale to a repository of one billion files, 35 million commits, and tens of thousands of developers.
- Benefits include unified versioning, extensive code sharing, simplified dependency management, atomic changes, large-scale refactoring, collaboration across teams, flexible code ownership, and code visibility.
- Drawbacks include having to create and scale tools for development and execution and maintain code health, as well as potential for codebase complexity (such as unnecessary dependencies).



two billion lines of code in nine million unique source files. The total number of files also includes source files copied into release branches, files that are deleted at the latest revision, configuration files, documentation, and supporting data files; see the table here for a summary of Google's repository statistics from January 2015.

In 2014, approximately 15 million lines of code were changed<sup>b</sup> in approximately 250,000 files in the Google repository on a weekly basis. The Linux kernel is a prominent example of a large open source software repository containing approximately 15 million lines of code in 40,000 files.<sup>14</sup>

Google's codebase is shared by more

than 25,000 Google software developers from dozens of offices in countries around the world. On a typical workday, they commit 16,000 changes to the codebase, and another 24,000 changes are committed by automated systems. Each day the repository serves billions of file read requests, with approximately 800,000 queries per second during peak traffic and an average of approximately 500,000 queries per second each workday. Most of this traffic originates from Google's distributed build-and-test systems.<sup>c</sup>

Figure 2 reports the number of unique human committers per week to the main repository, January 2010–July 2015. Figure 3 reports commits per week to Google's main repository over the same time period. The line for total commits includes data for

both the interactive use case, or human users, and automated use cases. Larger dips in both graphs occur during holidays affecting a significant number of employees (such as Christmas Day and New Year's Day, American Thanksgiving Day, and American Independence Day).

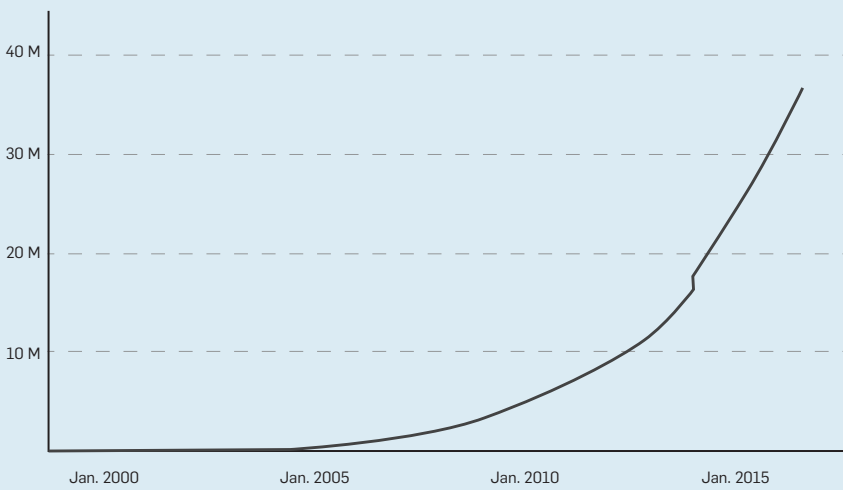
In October 2012, Google's central repository added support for Windows and Mac users (until then it was Linux-only), and the existing Windows and Mac repository was merged with the main repository. Google's tooling for repository merges attributes all historical changes being merged to their original authors, hence the corresponding bump in the graph in Figure 2. The effect of this merge is also apparent in Figure 1.

The commits-per-week graph shows the commit rate was dominated by human users until 2012, at which point Google switched to a custom-

<sup>b</sup> Includes only reviewed and committed code and excludes commits performed by automated systems, as well as commits to release branches, data files, generated files, open source files imported into the repository, and other non-source-code files.

<sup>c</sup> Google open sourced a subset of its internal build system; see <http://www.bazel.io>

**Figure 1. Millions of changes committed to Google's central repository over time.**



**Google repository statistics, January 2015.**

|                        |                    |
|------------------------|--------------------|
| Total number of files  | 1 billion          |
| Number of source files | 9 million          |
| Lines of source code   | 2 billion          |
| Depth of history       | 35 million commits |
| Size of content        | 86TB               |
| Commits per workday    | 40,000             |

source-control implementation for hosting the central repository, as discussed later. Following this transition, automated commits to the repository began to increase. Growth in the commit rate continues primarily due to automation.

Managing this scale of repository and activity on it has been an ongoing challenge for Google. Despite several years of experimentation, Google was not able to find a commercially available or open source version-control system to support such scale in a single repository. The Google proprietary system that was built to store, version, and vend this codebase is code-named Piper.

### Background

Before reviewing the advantages and disadvantages of working with a monolithic repository, some background on Google's tooling and workflows is needed.

**Piper and CitC.** Piper stores a single large repository and is implemented on top of standard Google infrastructure, originally Bigtable,<sup>2</sup> now Spanner.<sup>3</sup> Piper is distributed over 10 Google data centers around the world, relying on the Paxos<sup>6</sup> algorithm to guarantee consistency across replicas. This architecture provides a high level of redundancy and helps optimize latency for Google software developers, no matter where they work. In addition, caching and asynchronous operations hide much of the network latency from developers. This is important because gaining the full benefit of Google's cloud-based toolchain requires developers to be online.

Google relied on one primary Perforce instance, hosted on a single machine, coupled with custom caching infrastructure<sup>1</sup> for more than 10 years prior to the launch of Piper. Continued scaling of

**Figure 2. Human committers per week.**

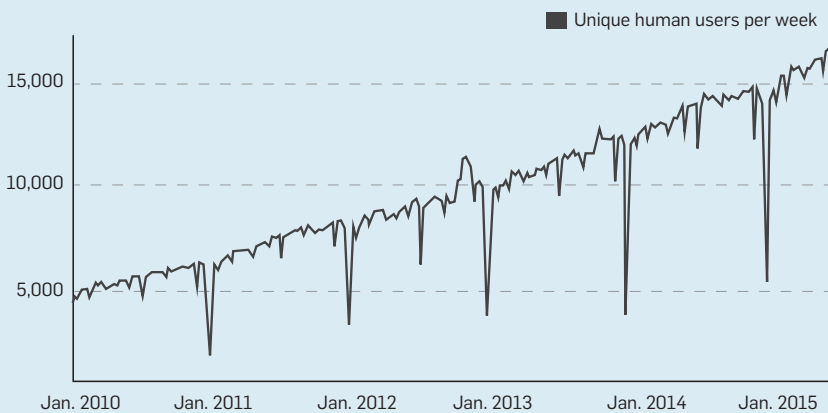
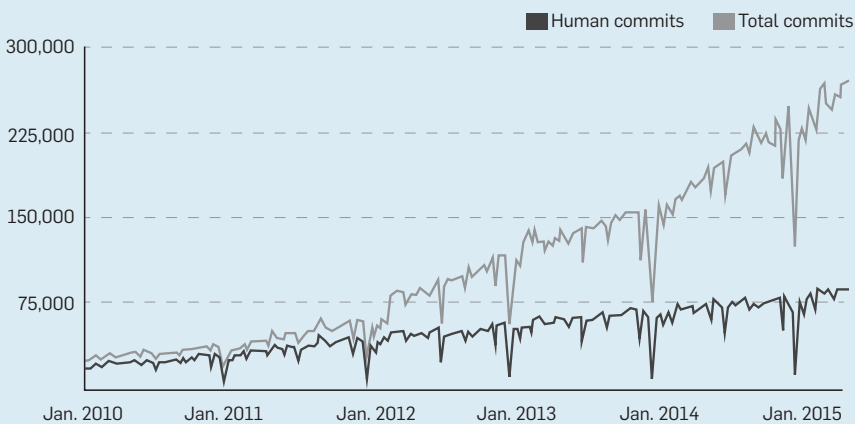


**Figure 3. Commits per week.**



the Google repository was the main motivation for developing Piper.

Since Google's source code is one of the company's most important assets, security features are a key consideration in Piper's design. Piper supports file-level access control lists. Most of the repository is visible to all Piper users;<sup>d</sup> however, important configuration files or files including business-critical algorithms can be more tightly controlled. In addition, read and write access to files in Piper is logged. If sensitive data is accidentally committed to Piper, the file in question can be purged. The read logs allow administrators to determine if anyone accessed the problematic file before it was removed.

In the Piper workflow (see Figure 4), developers create a local copy of files in the repository before changing them. These files are stored in a workspace owned by the developer. A Piper workspace is comparable to a working copy in Apache Subversion, a local clone in Git, or a client in Perforce. Updates from the Piper repository can be pulled into a workspace and merged with ongoing work, as desired (see Figure 5). A snapshot of the workspace can be shared with other developers for review. Files in a workspace are committed to the central repository only after going through the Google code-review process, as described later.

Most developers access Piper through a system called Clients in the Cloud, or CitC, which consists of a cloud-based storage backend and a Linux-only FUSE<sup>13</sup> file system. Developers see their workspaces as directories in the file system, including their changes overlaid on top of the full Piper repository. CitC supports code browsing and normal Unix tools with no need to clone or sync state locally. Developers can browse and edit files anywhere across the Piper repository, and only modified files are stored in their workspace. This structure means CitC workspaces typically consume only a small amount of storage (an average workspace has fewer than 10 files) while presenting a seamless view of the entire Piper codebase to the developer.

All writes to files are stored as snapshots in CitC, making it possible to recover previous stages of work as needed. Snapshots may be explicitly named, restored, or tagged for review.

CitC workspaces are available on any machine that can connect to the cloud-based storage system, making it easy to switch machines and pick up work without interruption. It also makes it possible for developers to view each other's work in CitC workspaces. Storing all in-progress work in the cloud is an important element of the Google workflow process. Working state is thus available to other tools, including the cloud-based build system, the automated test infrastructure, and the code browsing, editing, and review tools.

Several workflows take advantage of the availability of uncommitted code in CitC to make software developers working with the large codebase more productive. For instance, when sending a change out for code review, developers can enable an auto-commit option, which is particularly useful when code authors and reviewers are in different time zones. When the review is marked as complete, the tests will run; if they pass, the code will be committed to the repository without further human intervention. The Google code-browsing tool CodeSearch supports simple edits using CitC workspaces. While browsing the repository, developers can click on a button to enter edit mode and make a simple change (such as fixing a typo or improving

Figure 4. Piper workflow.



Figure 5. Piper team logo "Piper is Piper expanded recursively;" design source: Kirrily Anderson.



<sup>d</sup> Over 99% of files stored in Piper are visible to all full-time Google engineers.

a comment). Then, without leaving the code browser, they can send their changes out to the appropriate reviewers with auto-commit enabled.

Piper can also be used without CitC. Developers can instead store Piper workspaces on their local machines. Piper also has limited interoperability with Git. Over 80% of Piper users today use CitC, with adoption continuing to grow due to the many benefits provided by CitC.

Piper and CitC make working productively with a single, monolithic source repository possible at the scale of the Google codebase. The design and architecture of these systems were both heavily influenced by the trunk-based development paradigm employed at Google, as described here.

**Trunk-based development.** Google practices trunk-based development on top of the Piper source repository. The vast majority of Piper users work at the “head,” or most recent, version of a single copy of the code called “trunk” or “mainline.” Changes are made to the repository in a single, serial ordering. The combination of trunk-based development with a central repository defines the monolithic codebase model. Immediately after any commit, the new code is visible to, and usable by, all other developers. The fact that Piper users work on a single consistent view of the Google codebase is key for providing the advantages described later in this article.

Trunk-based development is beneficial in part because it avoids the painful merges that often occur when it is time to reconcile long-lived branches. Development on branches is unusual and not well supported at Google, though branches are typically used for releases. Release branches are cut from a specific revision of the repository. Bug fixes and enhancements that must be added to a release are typically developed on mainline, then cherry-picked into the release branch (see Figure 6). Due to the need to maintain stability and limit churn on the release branch, a release is typically a snapshot of head, with an optional small number of cherry-picks pulled in from head as needed. Use of long-lived branches with parallel development on the branch and mainline is exceedingly rare.



## Piper and CitC make working productively with a single, monolithic source repository possible at the scale of the Google codebase.



When new features are developed, both new and old code paths commonly exist simultaneously, controlled through the use of conditional flags. This technique avoids the need for a development branch and makes it easy to turn on and off features through configuration updates rather than full binary releases. While some additional complexity is incurred for developers, the merge problems of a development branch are avoided. Flag flips make it much easier and faster to switch users off new implementations that have problems. This method is typically used in project-specific code, not common library code, and eventually flags are retired so old code can be deleted. Google uses a similar approach for routing live traffic through different code paths to perform experiments that can be tuned in real time through configuration changes. Such A/B experiments can measure everything from the performance characteristics of the code to user engagement related to subtle product changes.

**Google workflow.** Several best practices and supporting systems are required to avoid constant breakage in the trunk-based development model, where thousands of engineers commit thousands of changes to the repository on a daily basis. For instance, Google has an automated testing infrastructure that initiates a rebuild of all affected dependencies on almost every change committed to the repository. If a change creates widespread build breakage, a system is in place to automatically undo the change. To reduce the incidence of bad code being committed in the first place, the highly customizable Google “presubmit” infrastructure provides automated testing and analysis of changes before they are added to the codebase. A set of global presubmit analyses are run for all changes, and code owners can create custom analyses that run only on directories within the codebase they specify. A small set of very low-level core libraries uses a mechanism similar to a development branch to enforce additional testing before new versions are exposed to client code.

An important aspect of Google culture that encourages code quality is the expectation that all code is reviewed

before being committed to the repository. Most developers can view and propose changes to files anywhere across the entire codebase—with the exception of a small set of highly confidential code that is more carefully controlled. The risk associated with developers changing code they are not deeply familiar with is mitigated through the code-review process and the concept of code ownership. The Google codebase is laid out in a tree structure. Each and every directory has a set of owners who control whether a change to files in their directory will be accepted. Owners are typically the developers who work on the projects in the directories in question. A change often receives a detailed code review from one developer, evaluating the quality of the change, and a commit approval from an owner, evaluating the appropriateness of the change to their area of the codebase.

Code reviewers comment on aspects of code quality, including design, functionality, complexity, testing, naming, comment quality, and code style, as documented by the various language-specific Google style guides.<sup>e</sup> Google has written a code-review tool called Critique that allows the reviewer to view the evolution of the code and comment on any line of the change. It encourages further revisions and a conversation leading to a final “Looks Good To Me” from the reviewer, indicating the review is complete.

Google’s static analysis system (Tricorder<sup>10</sup>) and presubmit infrastructure also provide data on code quality, test coverage, and test results automatically in the Google code-review tool. These computationally intensive checks are triggered periodically, as well as when a code change is sent for review. Tricorder also provides suggested fixes with one-click code editing for many errors. These systems provide important data to increase the effectiveness of code reviews and keep the Google codebase healthy.

A team of Google developers will occasionally undertake a set of wide-reaching code-cleanup changes to further maintain the health of the codebase. The developers who perform these changes commonly separate

them into two phases. With this approach, a large backward-compatible change is made first. Once it is complete, a second smaller change can be made to remove the original pattern that is no longer referenced. A Google tool called Rosie<sup>f</sup> supports the first phase of such large-scale cleanups and code changes. With Rosie, developers create a large patch, either through a find-and-replace operation across the entire repository or through more complex refactoring tools. Rosie then takes care of splitting the large patch into smaller patches, testing them independently, sending them out for code review, and committing them automatically once they pass tests and a code review. Rosie splits patches along project directory lines, relying on the code-ownership hierarchy described earlier to send patches to the appropriate reviewers.

Figure 7 reports the number of changes committed through Rosie on a monthly basis, demonstrating the importance of Rosie as a tool for

performing large-scale code changes at Google. Using Rosie is balanced against the cost incurred by teams needing to review the ongoing stream of simple changes Rosie generates. As Rosie’s popularity and usage grew, it became clear some control had to be established to limit Rosie’s use to high-value changes that would be distributed to many reviewers, rather than to single atomic changes or rejected. In 2013, Google adopted a formal large-scale change-review process that led to a decrease in the number of commits through Rosie from 2013 to 2014. In evaluating a Rosie change, the review committee balances the benefit of the change against the costs of reviewer time and repository churn. We later examine this and similar trade-offs more closely.

In sum, Google has developed a number of practices and tools to support its enormous monolithic codebase, including trunk-based development, the distributed source-code repository Piper, the workspace client CitC, and workflow-support-tools Critique, CodeSearch, Tricorder, and Rosie. We discuss the pros and cons of this model here.

<sup>f</sup> The project name was inspired by Rosie the robot maid from the TV series “The Jetsons.”

**Figure 6. Release branching model.**

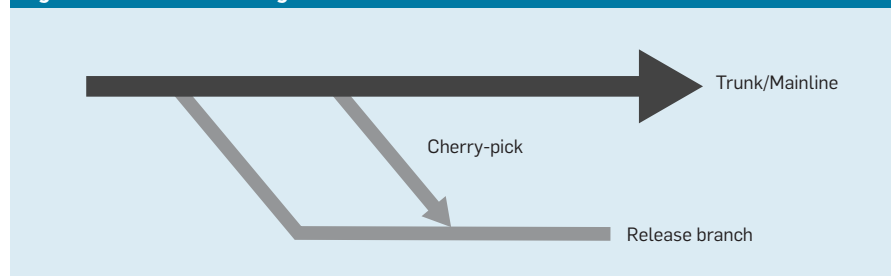
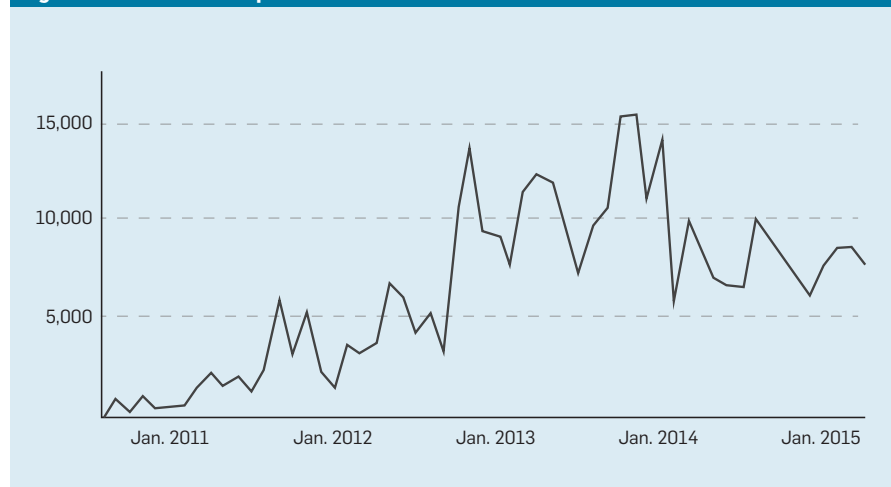


**Figure 7. Rosie commits per month.**



<sup>e</sup> <https://github.com/google/styleguide>

## Analysis

This section outlines and expands upon both the advantages of a monolithic codebase and the costs related to maintaining such a model at scale.

**Advantages.** Supporting the ultra-large-scale of Google's codebase while maintaining good performance for tens of thousands of users is a challenge, but Google has embraced the monolithic model due to its compelling advantages.

Most important, it supports:

- ▶ Unified versioning, one source of truth;
- ▶ Extensive code sharing and reuse;
- ▶ Simplified dependency management;
- ▶ Atomic changes;
- ▶ Large-scale refactoring;
- ▶ Collaboration across teams;
- ▶ Flexible team boundaries and code ownership; and
- ▶ Code visibility and clear tree structure providing implicit team namespacing.

A single repository provides unified versioning and a single source of truth. There is no confusion about which repository hosts the authoritative version of a file. If one team wants to depend on another team's code, it can depend on it directly. The Google codebase includes a wealth of useful libraries, and the monolithic repository leads to extensive code sharing and reuse.

The Google build system<sup>5</sup> makes it easy to include code across directories, simplifying dependency management. Changes to the dependencies of a project trigger a rebuild of the dependent code. Since all code is ver-

sioned in the same repository, there is only ever one version of the truth, and no concern about independent versioning of dependencies.

Most notably, the model allows Google to avoid the “diamond dependency” problem (see Figure 8) that occurs when A depends on B and C, both B and C depend on D, but B requires version D.1 and C requires version D.2. In most cases it is now impossible to build A. For the base library D, it can become very difficult to release a new version without causing breakage, since all its callers must be updated at the same time. Updating is difficult when the library callers are hosted in different repositories.

In the open source world, dependencies are commonly broken by library updates, and finding library versions that all work together can be a challenge. Updating the versions of dependencies can be painful for developers, and delays in updating create technical debt that can become very expensive. In contrast, with a monolithic source tree it makes sense, and is easier, for the person updating a library to update all affected dependencies at the same time. The technical debt incurred by dependent systems is paid down immediately as changes are made. Changes to base libraries are instantly propagated through the dependency chain into the final products that rely on the libraries, without requiring a separate sync or migration step.

Note the diamond-dependency problem can exist at the source/API level, as described here, as well as between binaries.<sup>12</sup> At Google, the

binary problem is avoided through use of static linking.

The ability to make atomic changes is also a very powerful feature of the monolithic model. A developer can make a major change touching hundreds or thousands of files across the repository in a single consistent operation. For instance, a developer can rename a class or function in a single commit and yet not break any builds or tests.

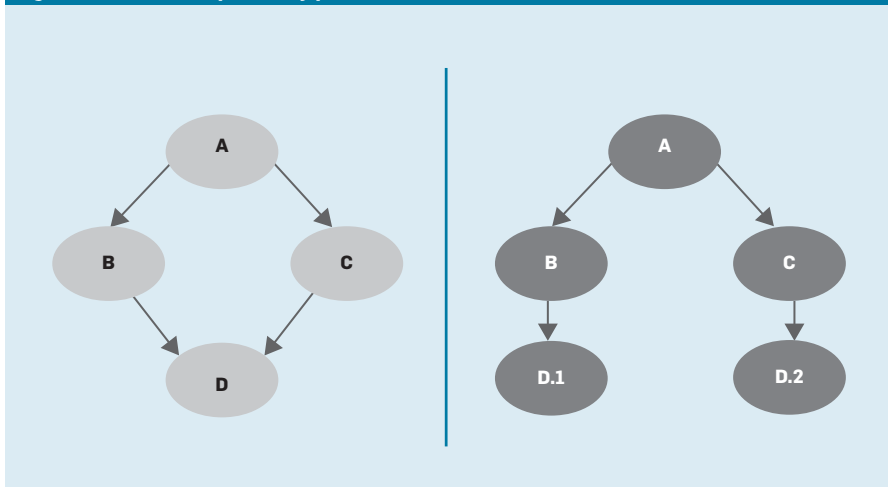
The availability of all source code in a single repository, or at least on a centralized server, makes it easier for the maintainers of core libraries to perform testing and performance benchmarking for high-impact changes before they are committed. This approach is useful for exploring and measuring the value of highly disruptive changes. One concrete example is an experiment to evaluate the feasibility of converting Google data centers to support non-x86 machine architectures.

With the monolithic structure of the Google repository, a developer never has to decide where the repository boundaries lie. Engineers never need to “fork” the development of a shared library or merge across repositories to update copied versions of code. Team boundaries are fluid. When project ownership changes or plans are made to consolidate systems, all code is already in the same repository. This environment makes it easy to do gradual refactoring and reorganization of the codebase. The change to move a project and update all dependencies can be applied atomically to the repository, and the development history of the affected code remains intact and available.

Another attribute of a monolithic repository is the layout of the codebase is easily understood, as it is organized in a single tree. Each team has a directory structure within the main tree that effectively serves as a project's own namespace. Each source file can be uniquely identified by a single string—a file path that optionally includes a revision number. Browsing the codebase, it is easy to understand how any source file fits into the big picture of the repository.

The Google codebase is constantly evolving. More complex codebase modernization efforts (such as updat-

**Figure 8. Diamond dependency problem.**



ing it to C++11 or rolling out performance optimizations<sup>9)</sup> are often managed centrally by dedicated codebase maintainers. Such efforts can touch half a million variable declarations or function-call sites spread across hundreds of thousands of files of source code. Because all projects are centrally stored, teams of specialists can do this work for the entire company, rather than require many individuals to develop their own tools, techniques, or expertise.

As an example of how these benefits play out, consider Google's Compiler team, which ensures developers at Google employ the most up-to-date toolchains and benefit from the latest improvements in generated code and "debuggability." The monolithic repository provides the team with full visibility of how various languages are used at Google and allows them to do codebase-wide cleanups to prevent changes from breaking builds or creating issues for developers. This greatly simplifies compiler validation, thus reducing compiler release cycles and making it possible for Google to safely do regular compiler releases (typically more than 20 per year for the C++ compilers).

Using the data generated by performance and regression tests run on nightly builds of the entire Google codebase, the Compiler team tunes default compiler settings to be optimal. For example, due to this centralized effort, Google's Java developers all saw their garbage collection (GC) CPU consumption decrease by more than 50% and their GC pause time decrease by 10%–40% from 2014 to 2015. In addition, when software errors are discovered, it is often possible for the team to add new warnings to prevent recurrence. In conjunction with this change, they scan the entire repository to find and fix other instances of the software issue being addressed, before turning to new compiler errors. Having the compiler-reject patterns that proved problematic in the past is a significant boost to Google's overall code health.

Storing all source code in a common version-control repository allows codebase maintainers to efficiently analyze and change Google's source code. Tools like Refaster<sup>11</sup> and ClangMR<sup>15</sup>



**An important aspect of Google culture that encourages code quality is the expectation that all code is reviewed before being committed to the repository.**



(often used in conjunction with Rosie) make use of the monolithic view of Google's source to perform high-level transformations of source code. The monolithic codebase captures all dependency information. Old APIs can be removed with confidence, because it can be proven that all callers have been migrated to new APIs. A single common repository vastly simplifies these tools by ensuring atomicity of changes and a single global view of the entire repository at any given time.

**Costs and trade-offs.** While important to note a monolithic codebase in no way implies monolithic software design, working with this model involves some downsides, as well as trade-offs, that must be considered.

These costs and trade-offs fall into three categories:

- ▶ Tooling investments for both development and execution;
- ▶ Codebase complexity, including unnecessary dependencies and difficulties with code discovery; and
- ▶ Effort invested in code health.

In many ways the monolithic repository yields simpler tooling since there is only one system of reference for tools working with source. However, it is also necessary that tooling scale to the size of the repository. For instance, Google has written a custom plug-in for the Eclipse integrated development environment (IDE) to make working with a massive codebase possible from the IDE. Google's code-indexing system supports static analysis, cross-referencing in the code-browsing tool, and rich IDE functionality for Emacs, Vim, and other development environments. These tools require ongoing investment to manage the ever-increasing scale of the Google codebase.

Beyond the investment in building and maintaining scalable tooling, Google must also cover the cost of running these systems, some of which are very computationally intensive. Much of Google's internal suite of developer tools, including the automated test infrastructure and highly scalable build infrastructure, are critical for supporting the size of the monolithic codebase. It is thus necessary to make trade-offs concerning how frequently to run this tooling to balance the cost of execution vs. the benefit of the data provided to developers.

The monolithic model makes it easier to understand the structure of the codebase, as there is no crossing of repository boundaries between dependencies. However, as the scale increases, code discovery can become more difficult, as standard tools like `grep` bog down. Developers must be able to explore the codebase, find relevant libraries, and see how to use them and who wrote them. Library authors often need to see how their APIs are being used. This requires a significant investment in code search and browsing tools. However, Google has found this investment highly rewarding, improving the productivity of all developers, as described in more detail by Sadowski et al.<sup>9</sup>

Access to the whole codebase encourages extensive code sharing and reuse. Some would argue this model, which relies on the extreme scalability of the Google build system, makes it too easy to add dependencies and reduces the incentive for software developers to produce stable and well-thought-out APIs.

Due to the ease of creating dependencies, it is common for teams to not think about their dependency graph, making code cleanup more error-prone. Unnecessary dependencies can increase project exposure to downstream build breakages, lead to binary size bloating, and create additional work in building and testing. In addition, lost productivity ensues when abandoned projects that remain in the repository continue to be updated and maintained.

Several efforts at Google have sought to rein in unnecessary dependencies. Tooling exists to help identify and remove unused dependencies, or dependencies linked into the product binary for historical or accidental reasons, that are not needed. Tooling also exists to identify underutilized dependencies, or dependencies on large libraries that are mostly unneeded, as candidates for refactoring.<sup>7</sup> One such tool, Clipper, relies on a custom Java compiler to generate an accurate cross-reference index. It then uses the index to construct a reachability graph and determine what classes are never used. Clipper is useful in guiding dependency-refactoring efforts by finding targets that are relatively easy to remove or break up.



**A developer can make a major change touching hundreds or thousands of files across the repository in a single consistent operation.**



Dependency-refactoring and cleanup tools are helpful, but, ideally, code owners should be able to prevent unwanted dependencies from being created in the first place. In 2011, Google started relying on the concept of API visibility, setting the default visibility of new APIs to “private.” This forces developers to explicitly mark APIs as appropriate for use by other teams. A lesson learned from Google’s experience with a large monolithic repository is such mechanisms should be put in place as soon as possible to encourage more hygienic dependency structures.

The fact that most Google code is available to all Google developers has led to a culture where some teams expect other developers to read their code rather than providing them with separate user documentation. There are pros and cons to this approach. No effort goes toward writing or keeping documentation up to date, but developers sometimes read more than the API code and end up relying on underlying implementation details. This behavior can create a maintenance burden for teams that then have trouble deprecating features they never meant to expose to users.

This model also requires teams to collaborate with one another when using open source code. An area of the repository is reserved for storing open source code (developed at Google or externally). To prevent dependency conflicts, as outlined earlier, it is important that only one version of an open source project be available at any given time. Teams that use open source software are expected to occasionally spend time upgrading their codebase to work with newer versions of open source libraries when library upgrades are performed.

Google invests significant effort in maintaining code health to address some issues related to codebase complexity and dependency management. For instance, special tooling automatically detects and removes dead code, splits large refactorings and automatically assigns code reviews (as through Rosie), and marks APIs as deprecated. Human effort is required to run these tools and manage the corresponding large-scale code changes. A cost is also incurred

by teams that need to review an ongoing stream of simple refactorings resulting from codebase-wide clean-ups and centralized modernization efforts.

## Alternatives

As the popularity and use of distributed version control systems (DVCSs) like Git have grown, Google has considered whether to move from Piper to Git as its primary version-control system. A team at Google is focused on supporting Git, which is used by Google's Android and Chrome teams outside the main Google repository. The use of Git is important for these teams due to external partner and open source collaborations.

The Git community strongly suggests and prefers developers have more and smaller repositories. A Git-clone operation requires copying all content to one's local machine, a procedure incompatible with a large repository. To move to Git-based source hosting, it would be necessary to split Google's repository into thousands of separate repositories to achieve reasonable performance. Such reorganization would necessitate cultural and workflow changes for Google's developers. As a comparison, Google's Git-hosted Android codebase is divided into more than 800 separate repositories.

Given the value gained from the existing tools Google has built and the many advantages of the monolithic codebase structure, it is clear that moving to more and smaller repositories would not make sense for Google's main repository. The alternative of moving to Git or any other DVCS that would require repository splitting is not compelling for Google.

Current investment by the Google source team focuses primarily on the ongoing reliability, scalability, and security of the in-house source systems. The team is also pursuing an experimental effort with Mercurial,<sup>g</sup> an open source DVCS similar to Git. The goal is to add scalability features to the Mercurial client so it can efficiently support a codebase the size of Google's. This would provide Google's developers with an alternative of using popular DVCS-style workflows in conjunction with the central

repository. This effort is in collaboration with the open source Mercurial community, including contributors from other companies that value the monolithic source model.

## Conclusion

Google chose the monolithic-source-management strategy in 1999 when the existing Google codebase was migrated from CVS to Perforce. Early Google engineers maintained that a single repository was strictly better than splitting up the codebase, though at the time they did not anticipate the future scale of the codebase and all the supporting tooling that would be built to make the scaling feasible.

Over the years, as the investment required to continue scaling the centralized repository grew, Google leadership occasionally considered whether it would make sense to move from the monolithic model. Despite the effort required, Google repeatedly chose to stick with the central repository due to its advantages.

The monolithic model of source code management is not for everyone. It is best suited to organizations like Google, with an open and collaborative culture. It would not work well for organizations where large parts of the codebase are private or hidden between groups.

At Google, we have found, with some investment, the monolithic model of source management can scale successfully to a codebase with more than one billion files, 35 million commits, and thousands of users around the globe. As the scale and complexity of projects both inside and outside Google continue to grow, we hope the analysis and workflow described in this article can benefit others weighing decisions on the long-term structure for their codebases.

## Acknowledgments

We would like to recognize all current and former members of the Google Developer Infrastructure teams for their dedication in building and maintaining the systems referenced in this article, as well as the many people who helped in reviewing the article; in particular: Jon Perkins and Ingo Walther, the current Tech Leads of Piper; Kyle Lippincott and Crutcher Dunnivant, the current and former

Tech Leads of CitC; Hyrum Wright, Google's large-scale refactoring guru; and Chris Colohan, Caitlin Sadowski, Morgan Ames, Rob Siemborski, and the Piper and CitC development and support teams for their insightful review comments. 

## References

1. Bloch, D. *Still All on One Server: Perforce at Scale*. Google White Paper, 2011; <http://info.perforce.com/rs/perforce/images/GoogleWhitePaper-StillAllOnOneServer-PerforceatScale.pdf>
2. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R.E. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems* 26, 2 (June 2008).
3. Corbett, J.C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J., Ghemawat, S., Gubarev, A., Heiser, C., Hochschild, P. et al. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems* 31, 3 (Aug. 2013).
4. Gabriel, R.P., Northrop, L., Schmidt, D.C., and Sullivan, K. Ultra-large-scale systems. In *Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications* (Portland, OR, Oct. 22–26). ACM Press, New York, 2006, 632–634.
5. Kemper, C. Build in the Cloud: How the Build System works. Google Engineering Tools blog post, 2011; <http://google-engtools.blogspot.com/2011/08/build-in-cloud-how-build-system-works.html>
6. Lamport, L. Paxos made simple. *ACM Sigact News* 32, 4 (Nov. 2001), 18–25.
7. Morgenthaler, J.D., Gridnev, M., Sautciuc, R., and Bhansali, S. Searching for build debt: Experiences managing technical debt at Google. In *Proceedings of the Third International Workshop on Managing Technical Debt* (Zürich, Switzerland, June 2–9). IEEE Press Piscataway, NJ, 2012, 1–6.
8. Ren, G., Tune, E., Moseley, T., Shi, Y., Rus, S., and Hundt, R. Google-wide profiling: A continuous profiling infrastructure for data centers. *IEEE Micro* 30, 4 (2010), 65–79.
9. Sadowski, C., Stolee, K., and Elbaum, S. How developers search for code: A case study. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy, Aug. 30–Sept. 4). ACM Press, New York, 2015, 191–201.
10. Sadowski, C., van Gogh, J., Jaspán, C., Soederberg, E., and Winter, C. Tricorder: Building a program analysis ecosystem. In *Proceedings of the 37th International Conference on Software Engineering*, Vol. 1 (Firenze, Italy, May 16–24). IEEE Press Piscataway, NJ, 2015, 598–608.
11. Wasserman, L. Scalable, example-based refactorings with Refaster. In *Proceedings of the 2013 ACM Workshop on Refactoring Tools* (Indianapolis, IN, Oct. 26–31). ACM Press, New York, 2013, 25–28.
12. Wikipedia. Dependency hell. Accessed Jan. 20, 2015; [http://en.wikipedia.org/w/index.php?title=Dependency\\_hell&oldid=634636715](http://en.wikipedia.org/w/index.php?title=Dependency_hell&oldid=634636715)
13. Wikipedia. Filesystem in userspace. Accessed June 4, 2015; [http://en.wikipedia.org/w/index.php?title=Filesystem\\_in\\_userspace&oldid=664776514](http://en.wikipedia.org/w/index.php?title=Filesystem_in_userspace&oldid=664776514)
14. Wikipedia. Linux kernel. Accessed Jan. 20, 2015; [http://en.wikipedia.org/w/index.php?title=Linux\\_kernel&oldid=643170399](http://en.wikipedia.org/w/index.php?title=Linux_kernel&oldid=643170399)
15. Wright, H.K., Jasper, D., Klimek, M., Carruth, C., and Wan, Z. Large-scale automated refactoring using ClangMR. In *Proceedings of the IEEE International Conference on Software Maintenance* (Eindhoven, The Netherlands, Sept. 22–28). IEEE Press, 2013, 548–551.

Rachel Potvin (rpotvin@google.com) is an engineering manager at Google, Mountain View, CA.

Josh Levenberg (joshl@google.com) is a software engineer at Google, Mountain View, CA.

Copyright held by the authors

<sup>g</sup> <http://mercurial.selenic.com/>

DOI:10.1145/2851485

**The universal constant  $\lambda$ , the growth constant of polyominoes (think Tetris pieces), is rigorously proved to be greater than 4.**

BY GILL BAREQUET, GÜNTER ROTE, AND MIRA SHALAH

## $\lambda > 4$ An Improved Lower Bound on the Growth Constant of Polyominoes

What is  $\lambda$ ? The universal constant  $\lambda$  arises in the study of three completely unrelated fields: combinatorics, percolation, and branched polymers. In combinatorics, analysis of self-avoiding walks (SAWs, or non-self-intersecting lattice paths starting at the origin, counted by lattice units), simple polygons or self-avoiding polygons (SAPs, or closed SAWs, counted by either perimeter or area), and “polyominoes” (SAPs possibly with holes, or edge-connected sets of lattice squares, counted by area) are all related. In statistical physics, SAWs and SAPs play a significant role in percolation processes and in the collapse transition that branched polymers undergo when being heated. A collection edited by Guttmann<sup>15</sup> gives an excellent review of all

these topics and the connections between them. In this article, we describe our effort to prove that the growth constant (or asymptotic growth rate, also called the “connective constant”) of polyominoes is strictly greater than 4. To this aim, we exploited to the maximum possible computer resources

### » key insights

- Direct access to large shared RAM is useful for scientific computations, as well as for big-data applications.
- The growth constant of polyominoes is provably strictly greater than 4.
- The “proof” of this bound is an eigenvector of a giant matrix  $Q$  that can be verified as corresponding to the claimed bound—an eigenvalue of  $Q$ .





available to us. We eventually obtained a computer-generated proof that was verified by other programs implemented independently. We start with a brief introduction of the three research areas.

*Enumerative combinatorics.* Imagine a two-dimensional grid of square cells. A polyomino is a connected set of cells, where connectivity is along the sides of cells but not through corners; see Figure 1 for examples of small polyominoes. Polyominoes were popularized in the pioneering book by Golomb<sup>13</sup> and by Martin Gardner's columns in *Scientific American*; counting polyominoes by size became a popular and fascinating combinatorial problem. The size of a polyomino is the number of its cells.

Figure 2 shows a puzzle game with

polyominoes of sizes 5–8 cells. In this article, we consider “fixed” polyominoes; two such polyominoes are considered identical if one can be obtained from the other by a translation, while rotations and flipping are not allowed. The number of polyominoes of size  $n$  is usually denoted as  $A(n)$ . No formula is known for this number, but researchers have suggested efficient backtracking<sup>26,27</sup> and transfer-matrix<sup>9,18</sup> algorithms for computing  $A(n)$  for a given value of  $n$ . The latter algorithm was adapted in Barequet et al.<sup>5</sup> and also in this work for polyominoes on so-called twisted cylinders.

To date, the sequence  $A(n)$  has been determined up to  $n = 56$  by a parallel computation on a Hewlett Packard

server cluster using 64 processors.<sup>18</sup> The exact value of the growth constant of the sequence  $\lambda = \lim_{n \rightarrow \infty} A(n+1)/A(n)$  continues to be elusive. Incidentally, in the square lattice, the growth constant is very close to the “coordination number,” or the number of neighbors of each cell in the lattice (4 in our case). In this work we reveal (rigorously) the leading decimal digit of  $\lambda$ . It is 4. Moreover, we establish that  $\lambda$  is strictly greater than the coordination number.

*Percolation processes.* In physics, chemistry, and materials science, percolation theory deals with the movement and filtering of fluids through porous materials. Giving it a mathematical model, the theory describes the behavior of connected clusters in

**Figure 1.** The single monomino ( $A(1) = 1$ ), the two dominoes ( $A(2) = 2$ ), the  $A(3) = 6$  triominoes, and the  $A(4) = 19$  tetrominoes (Tetris pieces).

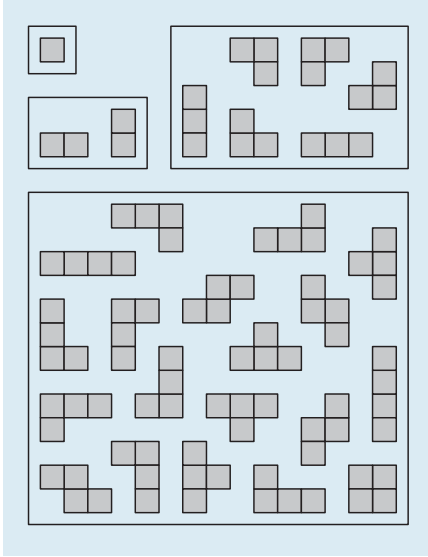
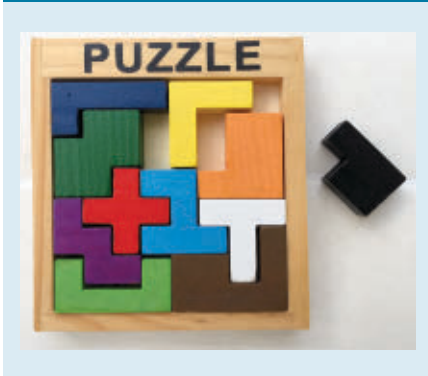


**Figure 2.** A solitaire game involving six polyominoes of size 5 (pentominoes), two of size 6 (hexominoes), two of size 7 (heptominoes), and one of size 8 (an octomino). The challenge is to tile the  $8 \times 8$  square box with the polyominoes.



random graphs. Suppose a unit of liquid  $L$  is poured on top of some porous material  $M$ . What is the chance  $L$  makes its way through  $M$  and reaches the bottom? An idealized mathematical model of this process is a two- or three-dimensional grid of vertices (“sites”) connected with edges (“bonds”), where each bond is independently open (or closed) for liquid flow with some probability  $\mu$ . In 1957, Broadbent and Hammersley<sup>8</sup> asked, for a fixed value of  $\mu$  and for the size of the grid tending to infinity, what is the probability that a path consisting of open bonds exists from the top to the bottom. They essentially investigated solute diffusing through solvent, molecules penetrat-

ing a porous solid, and similar processes, representing space as a lattice with two distinct types of cells.

In the literature of statistical physics, fixed polyominoes are usually called “strongly embedded lattice animals,” and in that context, the analogue of the growth rate of polyominoes is the growth constant of lattice animals. The terms high and low “temperature” mean high and low “density” of clusters, respectively, and the term “free energy” corresponds to the natural logarithm of the growth constant. Lattice animals were used for computing the mean cluster density in percolation processes, as in Gaunt et al.,<sup>12</sup> particularly in processes involving fluid flow in random media. Sykes and Glen<sup>29</sup> were the first to observe that  $A(n)$ , the total number of connected clusters of size  $n$ , grows asymptotically like  $C\lambda^n n^\theta$ , where  $\lambda$  is Klarner’s constant and  $C, \theta$  are two other fixed values.

*Collapse of branched polymers.* Another important topic in statistical physics is the existence of a collapse transition of branched polymers in dilute solution at a high temperature. In physics, a “field” is an entity each of whose points has a value that depends on location and time. Lubensky and Isaacson<sup>22</sup> developed a field theory of branched polymers in the dilute limit, using statistics of (bond) lattice animals (important in the theory of percolation) to imply when a solvent is good or bad for a polymer. Derrida and Herrmann<sup>10</sup> investigated two-dimensional branched polymers by looking at lattice animals on a square lattice and studying their free energy. Flesia et al.<sup>11</sup> made the connection between collapse processes to percolation theory, relating the growth constant of strongly embedded site animals to the free energy in the processes. Several models of branched polymers in dilute solution were considered by Madras et al.,<sup>24</sup> proving bounds on the growth constants for each such model.

*Brief History.* Determining the exact value of  $\lambda$  (or even setting good bounds on it) is a notably difficult problem in enumerative combinatorics. In 1967, Klarner<sup>19</sup> showed that the limit  $\lambda = \lim_{n \rightarrow \infty} A(n)^{1/n}$  exists. Since then,  $\lambda$  has been called “Klarner’s constant.” Only in 1999, Madras<sup>23</sup> proved the stronger statement that the asymptotic growth

rate in the sense of the limit  $\lambda = \lim_{n \rightarrow \infty} A(n+1)/A(n)$  exists.

By using interpolation methods, Sykes and Glen<sup>29</sup> estimated in 1976 that  $\lambda = 4.06 \pm 0.02$ . This estimate was refined several times based on heuristic extrapolation from the increasing number of known values of  $A(n)$ . The most accurate estimate— $4.0625696 \pm 0.0000005$ —was given by Jensen<sup>18</sup> in 2003. Before carrying out this project, the best proven bounds on  $\lambda$  were roughly 3.9801 from below<sup>5</sup> and 4.6496 from above.<sup>20</sup>  $\lambda$  has thus always been an elusive constant, of which not even a single significant digit was known. Our goal was to raise the lower bound on  $\lambda$  over the barrier of 4, and thus reveal its first decimal digit and prove that  $\lambda \neq 4$ . The current improvement of the lower bound on  $\lambda$  to 4.0025 also cuts the difference between the known lower bound and the estimated value of  $\lambda$  by approximately 25%—from 0.0825 to 0.0600.

## Computer-Assisted Proofs

Our proof relies heavily on computer calculations, thus raising questions about its credibility and reliability. There are two complementary approaches for addressing this issue: formally verified computations and certified computations.

*Formally verified computing.* In this paradigm, a program is accompanied by a correctness proof that is refined to such a degree of detail and formality it can be checked mechanically by a computer. This approach and, more generally, formally verified “mathematics,” has become feasible for industry-level software, as well as for mathematics far beyond toy problems due to big advances in recent years; see the review by Avigad and Harrison.<sup>2</sup> One highlight is the formally verified proof by Gonthier<sup>14</sup> of the Four-Color Theorem, whose original proof by Appel and Haken<sup>1</sup> in 1977 already relied on computer assistance and was the prime instance of discussion about the validity and appropriateness of computer methods in mathematical proofs. (Incidentally, one step in Gonthier’s proof also involves polyominoes.) Another example is the verification of Hales’s proof<sup>21</sup> of the Kepler conjecture, which states the face-centered cubic pack-

ing of spheres is the densest possible. This proof, in addition to involving extensive case enumerations at different levels, is also very complicated in the interaction between the various parts. In August 2014, a team headed by Hales announced the completion of the Flyspeck project, constructing a formal proof of Kepler's conjecture.<sup>16</sup> Yet another example is the proof by Tucker<sup>30</sup> for Lorenz's conjecture (number 14 in Smale's list of challenging problems for the 21<sup>st</sup> century). The conjecture states that Lorenz's system of three differential equations, providing a model for atmospheric convection, supports a strange attractor. Tucker (page 104)<sup>30</sup> described the run of a parallel ODE solver several times on different computer setups, obtaining similar results.

**Certified Computation.** This technique is based on the idea it may be easier to check a given answer for correctness than come up with such an answer from scratch. The prototype example is the solution of an equation like  $3x^3 - 4x^2 - 5x + 2 = 0$ . While it may be a challenge to find the solution  $x = 2$ , it is straightforward to substitute the solution into the equation and check whether it is fulfilled. The result is trustworthy not because it is accompanied by a formal proof but because it is so simple to check, much simpler than the algorithm (perhaps Newton iteration in this case) used to solve the problem in the first place. In addition to the solution itself, it may be required that a certifying algorithm provide a certificate in order to facilitate the checking procedure.<sup>25</sup> Developing such certificates and computing them without excessive overhead may require algorithmic innovations (see the first sidebar "Certified Computations").

In our case, the result computed by our program can be interpreted as the eigenvalue of a giant matrix  $Q$ , which is not explicitly stored but implicitly defined through an iteration procedure. The certificate is a vector  $v$  that is a good-enough approximation of the corresponding eigenvector. From this vector, one can compute certified bounds on the eigenvalue in a rather straightforward way by comparing the vector  $v$  to the product  $Qv$ .

## Certified Computations

A certifying algorithm not only produces the result but also justifies its correctness by supplying a certificate that makes it easy to check the result. In contrast with formally verified computation, correctness is established for a particular instance of the problem and a concrete result. Here, we illustrate this concept with a few examples; see the survey by McConnell et al.<sup>25</sup> for a thorough treatment.

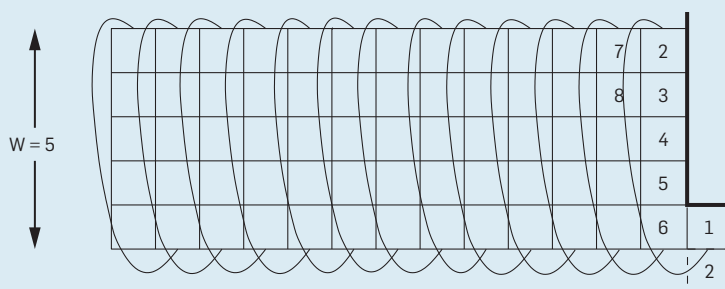
**The greatest common divisor.** The greatest common divisor of two numbers can be found through the ancient Euclidean algorithm. For example, the greatest common divisor of 880215 and 244035 is 15. Checking that 15 is indeed a common divisor is rather easy, but not clear is that 15 is the greatest. Luckily, the "extended" Euclidean algorithm provides a certificate: two integers  $p = -7571$  and  $q = 27308$ , such that  $880215p + 244035q = 15$ . This proves any common divisor of 880215 and 244035 must divide 15. No number greater than 15 can thus divide 880215 and 244035.

**Systems of linear equations and inequalities.** Consider the three equations  $4x - 3y + z = 2$ ,  $3x - y + z = 3$ , and  $x - 7y - z = 4$  in three unknowns  $x, y, z$ . It is straightforward to verify any proposed solution; however, the equations have no solution. Multiplying them by 4,  $-5$ ,  $-1$ , respectively, and adding them up leads to the contradiction  $0 = -11$ . The three multipliers thus provide an easy-to-check certificate for the answer. Such multipliers can always be found for an unsolvable linear system and can be computed as a by-product of the usual solution algorithms. A well-known extension of this example is linear programming, the optimization of a linear objective function subject to linear equations and inequalities, where an optimality certificate is provided by the dual solution.

**Testing a graph for 3-connectedness.** Certifying that a graph is not 3-connected is straightforward. The certificate consists of two vertices whose removal disconnects the graph into several pieces. It has been known since 1973 that a graph can be tested for 3-connectedness in linear time,<sup>17</sup> but all algorithms for this task are complicated. While providing a certificate in the negative case is easy, defining an easy-to-check certificate in the positive case and finding such a certificate in linear time has required graph-theoretic and algorithmic innovations.<sup>28</sup> This example illustrates that certifiability is not primarily an issue of running time. All algorithms, for constructing, as well as for checking, the certificate, run in linear time, just like classical non-certifying algorithms. The crucial distinction is that the short certificate-checking program is by far simpler than the construction algorithm.

**Comparison with the class NP.** There is some analogy between a certifying algorithm and the complexity class NP. For membership in NP, it is necessary only to have a certificate by which correctness of a solution can be checked in polynomial time. It does not matter how difficult it is to come up with the solution and find the certificate. In contrast with a certifying algorithm, the criterion for the checker is not simplicity but more clear-cut—running time. Another difference is that only "positive" answers to the problem need to be certified.

Figure 3. A twisted cylinder of perimeter  $W = 5$ .



These two approaches complement each other; a simpler checking procedure is more amenable to a formal proof and verification procedure. However, we did not go to such lengths; a formal verification of the program would have made sense only in the context of a full verification that includes also the human-readable

parts of the proof. We instead used traditional methods of ensuring program correctness of the certificate-checking program.

### Twisted Cylinders

A "twisted cylinder" is a half-infinite wraparound spiral-like square lattice (see Figure 3). We denote the perim-

Figure 4. A Motzkin path of length 7.

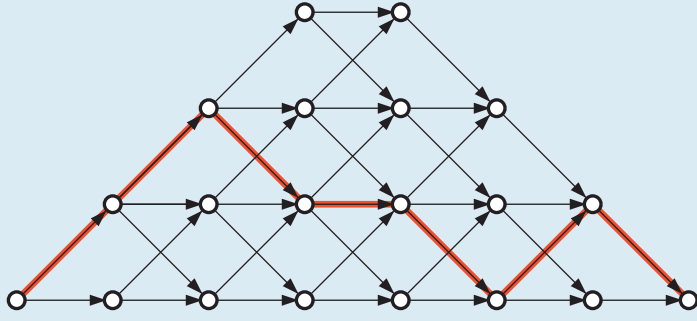
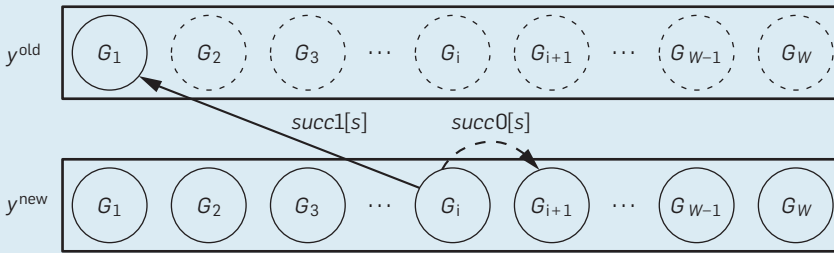


Figure 5. The dependence between the different groups of  $y^{\text{new}}$  and  $y^{\text{old}}$ .



eter or “width” of the twisted cylinder by  $W$ . Like in the plane, one can count polyominoes on a twisted cylinder of width  $W$  and study their growth constant,  $\lambda_W$ . Barequet et al. proved the sequence  $(\lambda_W)$  is monotone increasing<sup>5</sup> and converges to  $\lambda$ .<sup>3</sup> The bigger  $W$  is thus the better (higher) the lower bound  $\lambda_W$  on  $\lambda$  gets.

Analyzing the growth constant of polyominoes is more convenient on a twisted cylinder than in the plane. The reason is we want to build up polyominoes incrementally by considering one square at a time. On a twisted cylinder, this can be done in a uniform way, without having to jump to a new row from time to time. Imagine we walk along the spiral order of squares and at each square decide whether or not to add it to the polyomino. The size of a polyomino is the number of positive decisions we make along the way.

The crucial observation is no matter how big the polyominoes get, they can be characterized in a finite number of ways that depends only on  $W$ . All one needs to remember is the structure of the last  $W$  squares of the twisted cylinder (the “boundary” of the polyomino) and how they are interconnected through cells that were considered

before the boundary. This structure provides enough information for the continuation of the process; whenever a new square is considered and a decision is taken about whether or not to add it to the polyomino, the boundary is updated accordingly. This update is similar to the computation of connected components in a bi-level image by a row-wise scan. In this way, the growth of polyominoes on a twisted cylinder can be modeled by a finite-state automaton whose states are all possible boundaries. Every state of this automaton has two outgoing edges that correspond to whether or not the next square is added to the polyomino. A slight variation of this automaton can be seen in action in a video animation.<sup>7</sup> See the online appendix for more on automata for modeling polyominoes on twisted cylinders. See also the second sidebar “Representing Boundaries as Motzkin Paths.”

The number of states of the automaton that models the growth of polyominoes on a twisted cylinder of perimeter  $W$  is large<sup>4,5</sup>—the  $(W + 1)^{\text{st}}$  Motzkin number  $M_{W+1}$ . The  $n^{\text{th}}$  Motzkin number  $M_n$  counts Motzkin paths of length  $n$ ; see Figure 4 for an illustra-

tion of a Motzkin path and the sidebar for the relation between states and Motzkin paths. Such a path connects the integer grid points  $(0,0)$  and  $(n,0)$  with  $n$  steps, consisting only of steps taken from  $\{(1,1), (1,0), (1,-1)\}$  and not going under the  $x$  axis. Asymptotically,  $M_n \sim 3^n n^{-3/2}$ , and  $M_W$  thus increases roughly by a factor of 3 when  $W$  is incremented by 1.

The number of polyominoes with  $n$  cells that have state  $s$  as the boundary equals the number of paths the automaton can take from the starting state to  $s$ , involving  $n$  transitions in which a cell is added to the polyomino. We compute these numbers in a dynamic-programming recursion.

## Method

In 2004, a sequential program that computes  $\lambda_W$  for any perimeter was developed by Ares Ribó as part of her Ph.D. thesis under the supervision of author Günter Rote. The program first computes the endpoints of the outgoing edges from all states of the automaton and saves them in two long arrays *succ0* and *succ1* that correspond to adding an empty or an occupied cell. Both arrays are of length  $M := M_{W+1}$ . Two successive iteration vectors, which contain the number of polyominoes corresponding to each boundary, are stored as two arrays  $y^{\text{old}}$  and  $y^{\text{new}}$  of numbers, also of length  $M$ . The four arrays are indexed from 0 to  $M - 1$ . After initializing  $y^{\text{old}} := (1, 0, 0, \dots)$ , each iteration computes the new version of  $y$  through a very simple loop:

```

 $y^{\text{old}} := (1, 0, 0, \dots);$ 
repeat till convergence:
   $y^{\text{new}}[0] := 0;$ 
  for  $s := 1, \dots, M - 1:$ 
    (*)  $y^{\text{new}}[s] := y^{\text{new}}[\text{succ0}[s]]$ 
         $+ y^{\text{old}}[\text{succ1}[s]];$ 
   $y^{\text{old}} := y^{\text{new}};$ 
    
```

The pointer *succ0*[ $s$ ] may be null, in which case the corresponding zero entry ( $y^{\text{new}}[0]$ ) is used.

As explained earlier, each index  $s$  represents a state. The states are encoded by Motzkin paths, and these paths can be mapped to numbers  $s$  between 0 and  $M - 1$  in a bijective manner. See the online appendix for more.

In the iteration (\*), the vector  $y^{\text{new}}$  depends on itself, but this does not

cause any problem because  $\text{succ0}[s]$ , if it is non-null, is always less than  $s$ . There are thus no circular references, and each entry is set before it is used. In fact, the states can be partitioned into groups  $G_1, G_2, \dots, G_W$ ; the group  $G_i$  contains the states corresponding to boundaries in which  $i$  is the smallest index of an occupied cell, or the boundaries that start with  $i - 1$  empty cells. The dependence between the entries of the groups is shown schematically in Figure 5;  $\text{succ0}[s]$  of an entry  $s \in G_i$  (for  $1 \leq i \leq W - 1$ ), if it is non-null, belongs to  $G_{i+1}$ .

At the end,  $y^{\text{new}}$  is moved to  $y^{\text{old}}$  to start the new iteration. In the iteration (\*), the new vector  $y^{\text{new}}$  is a linear function of the old vector  $y^{\text{old}}$  and, as already indicated, can be written as a linear transformation  $y^{\text{old}} := Qy^{\text{old}}$ . The nonnegative integer matrix  $Q$  is implicitly given through the iteration (\*). We are interested in the growth rate of the sequence of vectors  $y^{\text{old}}$  that is determined by the dominant eigenvalue  $\lambda_w$  of  $Q$ . It is not difficult to show<sup>5</sup> that after every iteration,  $\lambda_w$  is contained in the interval

$$\min_s \frac{y^{\text{new}}[s]}{y^{\text{old}}[s]} \leq \lambda_w \leq \max_s \frac{y^{\text{new}}[s]}{y^{\text{old}}[s]}. \quad (1)$$

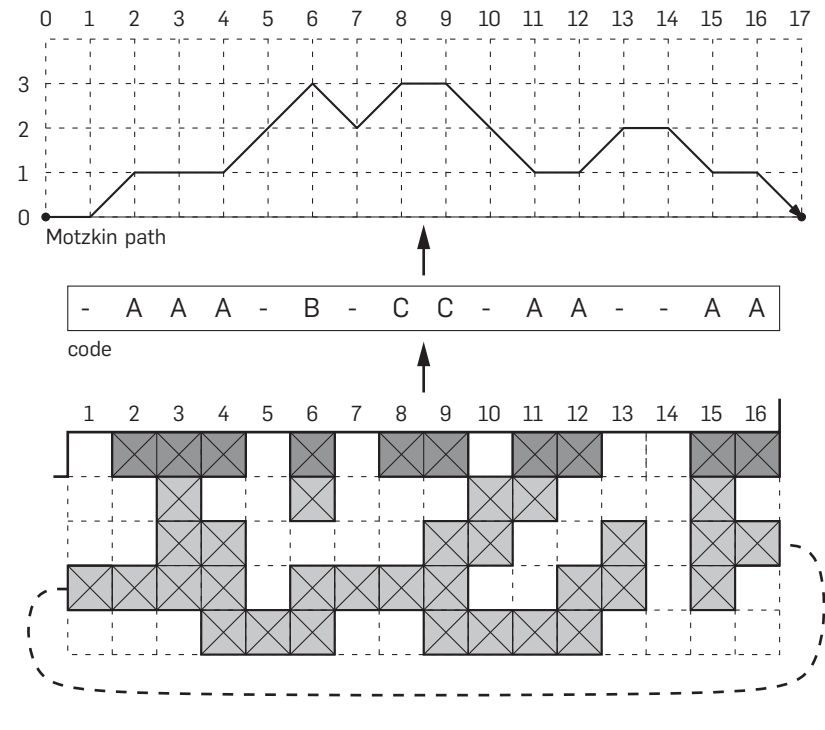
By the Perron-Frobenius Theorem about the eigenvalues of nonnegative matrices, the two bounds converge to  $\lambda_w$ , and  $y^{\text{old}}$  converges to a corresponding eigenvector.

As written, the program calculates the exact number of polyominoes of each size and state, provided the computations are carried out with precise integer arithmetic. However, these numbers grow like  $(\lambda_w)^n$ , and the program can thus not afford to store them exactly. Instead, we use single-precision floating-point numbers for  $y^{\text{old}}$  and  $y^{\text{new}}$ . Even so, the program must rescale the vector  $y^{\text{old}}$  from time to time in order to prevent floating-point overflow: Whenever the largest entry exceeds  $2^{80}$  at the end of an iteration, the program divides the whole vector by  $2^{100}$ . This rescaling does not affect the convergence of the process. The program terminates when the two bounds are close enough. The left-hand side of (1) is a lower bound on  $\lambda_w$ , which in turn is a lower bound on  $\lambda$ , and this is our real goal.

## Representing Boundaries as Motzkin Paths

The figure here illustrates the representation of polyomino boundaries as Motzkin paths. The bottom part of the figure shows a partially constructed polyomino on a twisted cylinder of width 16. The dashed line indicates two adjacent cells that are connected “around the cylinder,” where such a connection is not immediately apparent. The boundary cells (top row) are shown in darker gray. The light-gray cells away from the boundary need not be recorded individually; what matters is the connectivity among the boundary cells they provide. This connectivity is indicated in a symbolic code `-AAA-B-CC-AA--AA`. Boundary cells in the same component are represented by the same letter, and the character ‘-’ denotes an empty cell. However, we did not use this code in our program. Instead, we represented a boundary as a Motzkin path, as shown in the top part of the figure, because this representation allows for a convenient bijection to successive integers and thus for a compact storage of the boundary in a vector. Intuitively, the Motzkin path follows the movements of a stack when reading the code from left to right. Whenever a new component starts (such as component A in position 2 or component B in position 6), the path moves up. Whenever a component is temporarily interrupted (such as component A in position 5), the path also moves up. The path moves down when an interrupted component is resumed (such as component A in positions 11 and 15) or when a component is completed (positions 7, 10, and 17). The crucial property is that components cannot cross; that is, a pattern like `..A..B..A..B..` cannot occur. As a consequence of these rules, the occupied cells correspond to odd levels in the path, and the free cells correspond to even levels. The correspondence between boundaries and Motzkin paths was pointed out by Stefan Felsner of the Technische Universität Berlin (private communication).

Polyomino boundaries and Motzkin paths.



### Sequential Runs

In 2004, we obtained good approximations of  $y^{\text{old}}$  up to  $W = 22$ . The program required quite a bit of main memory (RAM) by the standards of the time. The computation of  $\lambda_{22} \approx$

3.9801 took approximately six hours on a single-processor machine with 32GB of RAM. (Today, the same program runs in 20 minutes on a regular workstation.) By extrapolating the first 22 values of the sequence  $\lambda_w$

(see Figure 6), we estimated that only when we reach  $W = 27$  would we break the mythical barrier of 4.0. However, as mentioned earlier, the storage requirement is proportional to  $M_{W_1}$  and  $M_W$  increases roughly by a factor of 3 when  $W$  is incremented by 1. With this exponential growth of both memory consumption and running time, the goal of breaking the barrier was then out of reach.

### Computing $\lambda_{27}$

*Environment.* In the spring of 2013, we were granted access to the Hewlett Packard ProLiant DL980 G7 server of the Hasso Plattner Institute Future SOC Lab in Potsdam, Germany (see Figure 7). This server consists of eight Intel Xeon X7560 nodes (Intel64 architecture), each with eight physical 2.26GHz processors (16 virtual cores), for a total of 64 processors (128 virtual

cores). Each node was equipped with 256GiB of RAM (and 24MiB of cache memory) for a total of 2TiB of RAM. Simultaneous access by all processors to the shared main memory was crucial to project success. Distributed memory would incur a severe penalty in running time. The machine ran under the Ubuntu version of the Gnu/Linux operating system. We used the C-compiler gcc with OpenMP 2.0 directives for parallel processing.

*Programming improvements.* Since for  $W = 27$  the finite automaton has  $M_{28} \approx 2.1 \cdot 10^{11}$  states, we initially estimated we would need memory for two 8-byte arrays (for storing *succ0* and *succ1*) and two 4-byte arrays (for storing  $y^{\text{old}}$  and  $y^{\text{new}}$ ), all of length  $M_{28}$ , for a total of  $24 \cdot 2.1 \cdot 10^{11} \approx 4.6$  TiB of RAM, which exceeded the server's available capacity. Only a combination of parallelization, storage-compression techniques,

and a few other enhancements allowed us to compute  $\lambda_{27}$  and push the lower bound on  $\lambda$  above 4.0. Here are the main memory-reduction tricks we used (see the online appendix for more):

*Elimination of unreachable states.* Approximately 11% of all states of the automaton are unreachable and do not affect the result;

*Bit-streaming of the succ0/1 arrays.* Instead of storing each entry of these arrays in a full word, we allocated just the required number of bits and stored the entries consecutively in a packed manner; in addition, we eliminated all the null *succ0*-pointers (approximately 11% of all pointers);

*Storing higher groups only once.* Approximately 50% of the states (those not in  $G_1$ ) were not needed in the recursion (\*); we thus did not keep in memory  $y^{\text{old}}[\cdot]$  of the respective entries;

*Recomputing succ0.* We computed the entries of the *succ0* array on-the-fly instead of keeping them in memory; and

*Streamlining the successor computation.* Instead of representing a Motzkin path by a sequence of  $W + 1$  integers, we kept a sequence of  $W + 1$  two-bit items we could store in one 8-byte word; this representation allowed us to use word-level operations and look-up tables for a fast mapping from paths to numbers.

To make full use of the parallel capacities, we used the partition of the set of states into groups, as in Figure 5, such that  $y^{\text{new}}$  of all elements in a group could be computed independently. We also parallelized the computation of the *succ* arrays in the preprocessing phase and various housekeeping tasks.

*Execution.* After 120 iterations, the program announced the lower bound 4.00064 on  $\lambda_{27}$ , thus breaking the 4 barrier. We continued to run the program for a few more days. On May 23, 2013, after 290 iterations, the program reached the stable situation (observed in a few successive tens of iterations)  $4.002537727 \leq \lambda_{27} \leq 4.002542973$ , establishing the new record  $\lambda > 4.00253$ . The total running time for the computations leading to this result was approximately 36 hours using approximately 1.5TiB of RAM. We had exclusive use of the server for a few dozen hours in total, spread over several weeks.

Figure 6. Extrapolating the sequence  $\lambda_W$ .

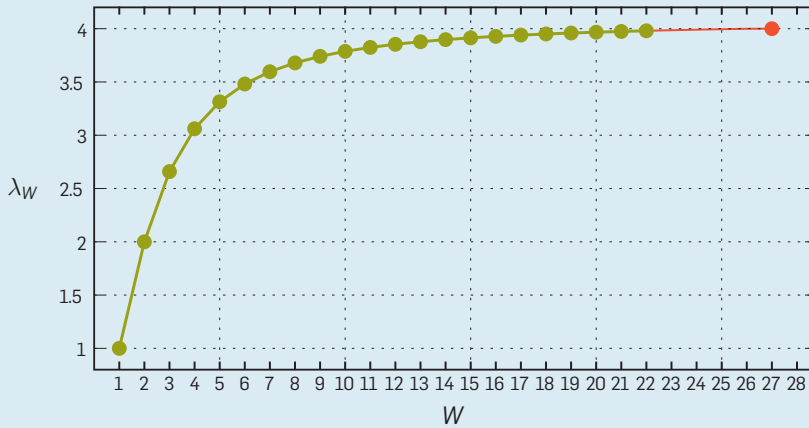


Figure 7. Front view of the “supercomputer” we used, a box of approximately 45×35×70 cm.



## Validity and Certification

Our proof depends heavily on computer calculations, raising two issues about its validity:

**Calculations.** Reproducing elaborate calculations on a large computer is difficult; particularly when a complicated parallel computer program is involved, everybody should be skeptical about the reliability of the results; and

**Computations.** We performed the computations with 32-bit floating-point numbers.

We address these issues in turn.

What our program was trying to compute is an eigenvalue of a matrix. The amount and length of the computations are irrelevant to the fact that eventually we have stored on disk a witness array of floating-point numbers (the “proof”), approximately 450GB in size, which is a good approximation of the eigenvector corresponding to  $\lambda_{27}$ . This array provides rigorous bounds on the true eigenvalue  $\lambda_{27}$ , because the relation (1) holds for any vector  $y^{\text{old}}$  and its successor vector  $y^{\text{new}}$ . To check the proof and evaluate the bounds (1), a program has to read only the approximate eigenvector  $y^{\text{old}}$  and carry out one iteration (\*). This approach of providing simple certificates for the result of complicated computations is the philosophy of “certifying algorithms.”<sup>25</sup>

To ensure the correctness of our checking program, we relied on traditional methods (such as testing and code inspection). Some parts of the program (such as reading the data from the files) are irrelevant for the correctness of the result. The main body of the program consists of a few simple loops (such as the iteration (\*) and the evaluation of (1)). The only technically challenging part of the algorithm is the successor computation. For this task, we had two programs at our disposal that were written independently by two people who used different state representations—and lived in different countries and did their work several years apart. We ran two different checking programs based on these procedures, giving us additional confidence. We also tested explicitly that the two successor programs yielded the same results. Both checking programs ran in a purely sequential manner, and the

running time was approximately 20 hours each.

Regarding the accuracy of the calculations, one can analyze how the recurrence (\*) produces  $y^{\text{new}}$  from  $y^{\text{old}}$ . One finds that each term in the lower bound (1) results from the input data (the approximate eigenvector  $y^{\text{old}}$ ) through at most 26 additions of positive numbers for computing  $y^{\text{new}}[s]$ , plus one division, all in single-precision float. The final minimization was error-free. Since we took care that no denormalized floating-point numbers occurred, the magnitude of the numerical errors was comparable to the accuracy of floating-point numbers, and the accumulated error was thus much smaller than the gap we opened between our new bound and 4. By carefully bounding the floating-point error, we obtained 4.00253176 as a certified lower bound on  $\lambda$ . In particular, we thus now know that the leading digit of  $\lambda$  is 4.

## Acknowledgments

We acknowledge the support of the facilities and staff of the Hasso Plattner Institute Future SOC Lab in Potsdam, Germany, who let us use their Hewlett Packard ProLiant DL980 G7 server. A technical version of this article<sup>6</sup> was presented at the 23<sup>rd</sup> Annual European Symposium on Algorithms in Patras, Greece, September 2015. 

## References

- Appel, K. and Haken, W. Every planar map is four colorable. *Illinois Journal of Mathematics* 21, 3 (Sept. 1977), 429–490 (part I) and 491–567 (part II).
- Avigad, J. and Harrison, J. Formally verified mathematics. *Commun. ACM* 57, 4 (April 2014), 66–75.
- Aleksandrowicz, G., Asinowski, A., Barequet, G., and Barequet, R. Formulae for polyominoes on twisted cylinders. In *Proceedings of the Eighth International Conference on Language and Automata Theory and Applications, Lecture Notes in Computer Science* 8370 (Madrid, Spain, Mar. 10–14). Springer-Verlag, Heidelberg, New York, Dordrecht, London, 2014, 76–87.
- Barequet, G. and Moffie, M. On the complexity of Jensen's algorithm for counting fixed polyominoes. *Journal of Discrete Algorithms* 5, 2 (June 2007), 348–355.
- Barequet, G., Moffie, M., Ribó, A., and Rote, G. Counting polyominoes on twisted cylinders. *INTEGERS: Electronic Journal of Combinatorial Number Theory* 6 (Sept. 2006), #A22, 37.
- Barequet, G., Rote, G., and Shalah, M.  $\lambda > 4$ . In *Proceedings of the 23<sup>rd</sup> Annual European Symposium on Algorithms Lecture Notes in Computer Science* 9294 (Patras, Greece, Sept. 14–16). Springer-Verlag, Berlin Heidelberg, Germany, 2015, 83–94.
- Barequet, G. and Shalah, M. Polyominoes on twisted cylinders. In the *Video Review at the 29<sup>th</sup> Annual Symposium on Computational Geometry* (Rio de Janeiro, Brazil, June 17–20). ACM Press, New York, 2013, 339–340; <http://www.computational-geometry.org/SoCG-videos/socg13video>
- Broadbent, S.R. and Hammersley, J.M. Percolation processes: I. Crystals and mazes. *Proceedings of the Cambridge Philosophical Society* 53, 3 (July 1957), 629–641.

- Conway, A. Enumerating 2D percolation series by the finite-lattice method: Theory. *Journal of Physics, A: Mathematical and General* 28, 2 (Jan. 1995), 335–349.
- Derrida, B. and Herrmann, H.J. Collapse of branched polymers. *Journal de Physique* 44, 12 (Dec. 1983), 1365–1376.
- Flesia, S., Gaunt, D.S., Soteros, C.E., and Whittington, S.G. Statistics of collapsing lattice animals. *Journal of Physics, A: Mathematical and General* 27, 17 (Sept. 1994), 5831–5846.
- Gaunt, D.S., Sykes, M.F., and Ruskin, H. Percolation processes in d-dimensions. *Journal of Physics A: Mathematical and General* 9, 11 (Nov. 1976), 1899–1911.
- Golomb, S.W. *Polyominoes, Second Edition*. Princeton University Press, Princeton, NJ, 1994.
- Gonthier, G. Formal proof—The four color theorem. *Notices of the AMS* 55, 11 (Dec. 2008), 1382–1393.
- Guttmann, A.J., Ed. *Polygons, Polyominoes and Polycubes, Lecture Notes in Physics* 775. Springer, Heidelberg, Germany, 2009.
- Hales, T., Solov'yev, A., and Le Truong, H. *The Flyspeck Project: Announcement of Completion*, Aug. 10, 2014; <https://code.google.com/p/flyspeck/wiki/AnnouncingCompletion>
- Hopcroft, J.E. and Tarjan, R.E. Dividing a graph into triconnected components. *SIAM Journal of Computing* 2, 3 (Sept. 1973), 135–158.
- Jensen, I. Counting polyominoes: A parallel implementation for cluster computing. In *Proceedings of the International Conference on Computational Science, Part III, Lecture Notes in Computer Science*, 2659 (Melbourne, Australia, and St. Petersburg, Russian Federation, June 2–4). Springer-Verlag, Berlin, Heidelberg, New York, 2003, 203–212.
- Klarner, D.A. Cell growth problems. *Canadian Journal of Mathematics* 19, 4 (1967), 851–863.
- Klarner, D.A. and Rivest, R.L. A procedure for improving the upper bound for the number of  $n$ -ominoes. *Canadian Journal of Mathematics* 25, 3 (Jan. 1973), 585–602.
- Lagarias, J.C., Ed. *The Kepler Conjecture: The Hales-Ferguson Proof*. Springer, New York, 2011.
- Lubensky, T.C. and Isaacson, J. Statistics of lattice animals and dilute branched polymers. *Physical Review A* 20, 5 (Nov. 1979), 2130–2146.
- Madras, N. A pattern theorem for lattice clusters. *Annals of Combinatorics* 3, 2–4 (June 1999), 357–384.
- Madras, N., Soteros, C.E., Whittington, S.G., Martin, J.L., Sykes, M.F., Flesia, S., and Gaunt, D.S. The free energy of a collapsing branched polymer. *Journal of Physics, A: Mathematical and General* 23, 22 (Nov. 1990), 5327–5350.
- McConnell, R.M., Mehlhorn, K., Näher, S., and Schweitzer, P. Certifying algorithms. *Computer Science Review* 5, 2 (May 2011), 119–161.
- Mertens, S. and Lautenbacher, M.E. Counting lattice animals: A parallel attack. *Journal of Statistical Physics* 66, 1–2 (Jan. 1992), 669–678.
- Redelmeier, D.H. Counting polyominoes: Yet another attack. *Discrete Mathematics* 36, 3 (Dec. 1981), 191–203.
- Schmidt, J.M. Contractions, removals and certifying 3-connectivity in linear time. *SIAM Journal on Computing* 42, 2 (Mar. 2013), 494–535.
- Sykes, M.F. and Glen, M. Percolation processes in two dimensions: I. Low-density series expansions. *Journal of Physics, A: Mathematical and General* 9, 1 (Jan. 1976), 87–95.
- Tucker, W. A rigorous ODE solver and Smale's 14<sup>th</sup> problem. *Foundations of Computational Mathematics* 2, 1 (Jan. 2002), 53–117.

**Gill Barequet** (barequet@cs.technion.ac.il) is an associate professor in and vice dean of the Department of Computer Science at the Technion - Israel Institute of Technology, Haifa, Israel.

**Günter Rote** (rote@inf.fu-berlin.de) is a professor in the Department of Computer Science at Freie Universität Berlin, Germany.

**Mira Shalah** (mshalah@cs.technion.ac.il) is a Ph.D. student, under the supervision of Gill Barequet, in the Department of Computer Science at the Technion - Israel Institute of Technology, Haifa, Israel.

**Today's social bots are sophisticated and sometimes menacing. Indeed, their presence can endanger online ecosystems as well as our society.**

BY EMILIO FERRARA, ONUR VAROL, CLAYTON DAVIS,  
FILIPPO MENCZER, AND ALESSANDRO FLAMMINI

# The Rise of Social Bots

BOTS (SHORT FOR software robots) have been around since the early days of computers. One compelling example of bots is chatbots, algorithms designed to hold a conversation with a human, as envisioned by Alan Turing in the 1950s.<sup>33</sup> The dream of designing a computer algorithm that passes the Turing test has driven artificial intelligence research for decades, as witnessed by initiatives like the Loebner Prize, awarding progress in natural language processing.<sup>a</sup> Many things have changed since the early days of AI, when bots like Joseph Weizenbaum's ELIZA,<sup>39</sup> mimicking a Rogerian psychotherapist, were developed as demonstrations or for delight.

Today, social media ecosystems populated by hundreds of millions of individuals present real incentives—including economic and political ones—

to design algorithms that exhibit human-like behavior. Such ecosystems also raise the bar of the challenge, as they introduce new dimensions to emulate in addition to content, including the social network, temporal activity, diffusion patterns, and sentiment expression. A social bot is a computer algorithm that automatically produces content and interacts with humans on social media, trying to emulate and possibly alter their behavior. Social bots have inhabited social media platforms for the past few years.<sup>7,24</sup>

## Engineered Social Tampering

What are the intentions of social bots? Some of them are benign and, in principle, innocuous or even helpful: this category includes bots that automatically aggregate content from various sources, like simple news feeds. Automatic responders to inquiries are increasingly adopted by brands and companies for customer care. Although these types of bots are designed to provide a useful service, they can sometimes be harmful, for example when they contribute to the spread of unverified information or rumors. Analyses of Twitter posts around the Boston marathon bombing revealed that social media can play an important role in the early recognition and characterization of emergency events.<sup>11</sup> But false accusations also circulated widely on Twitter in the

## » key insights

- **Social bots populate techno-social systems: they are often benign, or even useful, but some are created to harm, by tampering with, manipulating, and deceiving social media users.**
- **Social bots have been used to infiltrate political discourse, manipulate the stock market, steal personal information, and spread misinformation. The detection of social bots is therefore an important research endeavor.**
- **A taxonomy of the different social bot detection systems proposed in the literature accounts for network-based techniques, crowdsourcing strategies, feature-based supervised learning, and hybrid systems.**

a [www.loebner.net/Prizef/loebner-prize.html](http://www.loebner.net/Prizef/loebner-prize.html)



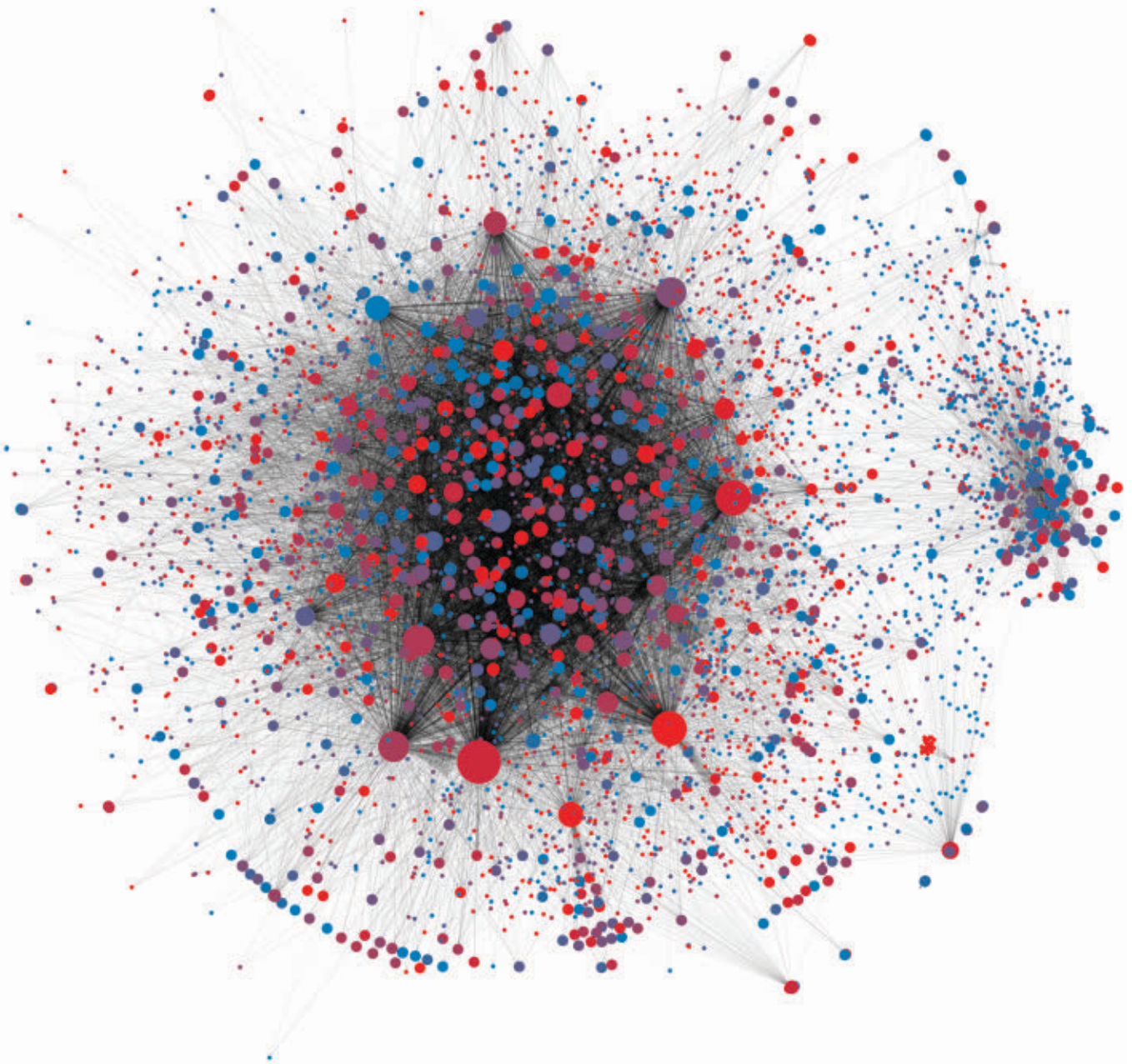
aftermath of the attack, mostly due to bots automatically retweeting posts without verifying the facts or checking the credibility of the source.<sup>20</sup>

With every new technology comes abuse, and social media is no exception. A second category of social bots includes malicious entities designed specifically with the purpose to harm. These bots mislead, exploit, and manipulate social media discourse with rumors, spam, malware, misinformation, slander, or even just noise. This

may result in several levels of damage to society. For example, bots may artificially inflate support for a political candidate;<sup>28</sup> such activity could endanger democracy by influencing the outcome of elections. In fact, this kind of abuse has already been observed: during the 2010 U.S. midterm elections, social bots were employed to support some candidates and smear their opponents, injecting thousands of tweets pointing to websites with fake news.<sup>28</sup> A similar case was report-

ed around the Massachusetts special election of 2010.<sup>26</sup> Campaigns of this type are sometimes referred to as *astroturf* or Twitter bombs.

The problem is not just establishing the veracity of the information being promoted—this was an issue before the rise of social bots, and remains beyond the reach of algorithmic approaches. The novel challenge brought by bots is the fact they can give the false impression that some piece of information, regardless of



**This network visualization illustrates how bots are used to affect, and possibly manipulate, the online debate about vaccination policy. It is the retweet network for the #SB277 hashtag, about a recent California law on vaccination requirements and exemptions. Nodes represent Twitter users, and links show how information spreads among users. The node size represents influence (times a user is retweeted), the color represents bot scores: red nodes are highly likely to be bot accounts, blue nodes are highly likely to be humans.**

its accuracy, is highly popular and endorsed by many, exerting an influence against which we haven't yet developed antibodies. Our vulnerability makes it possible for a bot to acquire significant influence, even unintentionally.<sup>2</sup> Sophisticated bots can generate personas that appear as credible followers, and thus are more difficult for both people and filtering algorithms to detect. They make for valuable entities on the fake follower market, and allegations of acquisition of fake followers have touched several prominent political figures in the U.S. and worldwide.

Journalists, analysts, and researchers increasingly report more examples of the potential dangers brought by social bots. These include the unwarranted consequences that the widespread diffusion of bots may have on the stability of markets. There have been claims that Twitter signals can be leveraged to predict the stock market,<sup>5</sup> and there is an increasing amount of evidence showing that market operators pay attention and react promptly to information from social media. On April 23, 2013, for example, the Syrian Electronic Army hacked the Twitter account of the Associated Press and posted a false rumor about a terror attack on the White House in which President Obama was allegedly injured. This provoked an immediate crash in the stock market. On May 6, 2010 a flash crash occurred in the U.S. stock market, when the Dow Jones plunged over 1,000 points (about 9%) within minutes—the biggest one-day point decline in history. After a five-month-long investigation, the role of high-frequency trading bots became obvious, but it yet remains unclear whether these bots had access to information from the social Web.<sup>22</sup>

The combination of social bots with an increasing reliance on automatic trading systems that, at least partially, exploit information from social media, is ripe with risks. Bots can amplify the visibility of misleading information, while automatic trading systems lack fact-checking capabilities. A recent orchestrated bot campaign successfully created the appearance of a sustained discussion about a tech company called Cynk. Automatic

trading algorithms picked up this conversation and started trading heavily in the company's stocks. This resulted in a 200-fold increase in market value, bringing the company's worth to \$5 billion.<sup>b</sup> By the time analysts recognized the orchestration behind this operation and stock trading was suspended, the losses were real.

### The Bot Effect

These anecdotes illustrate the consequences that tampering with the social Web may have for our increasingly interconnected society. In addition to potentially endangering democracy, causing panic during emergencies, and affecting the stock market, social bots can harm our society in even subtler ways. A recent study demonstrated the vulnerability of social media users to a social botnet designed to expose private information, like phone numbers and addresses.<sup>7</sup> This kind of vulnerability can be exploited by cybercrime and cause the erosion of trust in social media.<sup>22</sup> Bots can also hinder the advancement of public policy by creating the impression of a grassroots movement of contrarians, or contribute to the strong polarization of political discussion observed in social media.<sup>12</sup> They can alter the perception of social media influence, artificially enlarging the audience of some people,<sup>14</sup> or they can ruin the reputation of a company, for commercial or political purposes.<sup>25</sup> A recent study demonstrated that emotions are contagious on social media<sup>23</sup>; elusive bots could easily infiltrate a population of unaware humans and manipulate them to affect their perception of reality, with unpredictable results. Indirect social and economic effects of social bot activity include the alteration of social media analytics, adopted for various purposes such as TV ratings,<sup>c</sup> expert findings,<sup>40</sup> and scientific impact measurement.<sup>d</sup>

b The Curious Case of Cynk, an Abandoned Tech Company Now Worth \$5 Billion; mashable.com/2014/07/10/cynk

c Nielsen's New Twitter TV Ratings Are a Total Scam. Here's Why; defamer.gawker.com/niensens-new-twitter-tv-ratings-are-a-total-scam-here-1442214842

d altmetrics: a manifesto; altmetrics.org/manifesto/

### Act Like a Human, Think Like a Bot

One of the greatest challenges for bot detection in social media is in understanding what modern social bots can do.<sup>6</sup> Early bots mainly performed one type of activity: posting content automatically. These bots were naive and easy to spot by trivial detection strategies, such as focusing on high volume of content generation. In 2011, James Caverlee's team at Texas A&M University implemented a honeypot trap that managed to detect thousands of social bots.<sup>24</sup> The idea was simple and effective: the team created a few Twitter accounts (bots) whose role was solely to create nonsensical tweets with gibberish content, in which no human would ever be interested. However, these accounts attracted many followers. Further inspection confirmed that the suspicious followers were indeed social bots trying to grow their social circles by blindly following random accounts.

In recent years, Twitter bots have become increasingly sophisticated, making their detection more difficult. The boundary between human-like and bot-like behavior is now fuzzier. For example, social bots can search the Web for information and media to fill their profiles, and post collected material at predetermined times, emulating the human temporal signature of content production and consumption—including circadian patterns of daily activity and temporal spikes of information generation.<sup>19</sup> They can even engage in more complex types of interactions, such as entertaining conversations with other people, commenting on their posts, and answering their questions.<sup>22</sup> Some bots specifically aim to achieve greater influence by gathering new followers and expanding their social circles; they can search the social network for popular and influential people and follow them or capture their attention by sending them inquiries, in the hope to be noticed.<sup>2</sup> To acquire visibility, they can infiltrate popular discussions, generating topically appropriate—and even potentially interesting—content, by identifying relevant keywords and searching online for information fitting that conversation.<sup>17</sup> After the

appropriate content is identified, the bots can automatically produce responses through natural language algorithms, possibly including references to media or links pointing to external resources. Other bots aim at tampering with the identities of legitimate people: some are identity thieves, adopting slight variants of real usernames, and stealing personal information such as pictures and links. Even more advanced mechanisms can be employed; some social bots are able to “clone” the behavior of legitimate users, by interacting with their friends and posting topically coherent content with similar temporal patterns.

### A Taxonomy of Social Bot Detection Systems

For all the reasons outlined here, the computing community is engaging in the design of advanced methods to automatically detect social bots, or to discriminate between humans and bots. The strategies currently employed by social media services appear inadequate to contrast this phenomenon and the efforts of the academic community in this direction just started.

Here, we propose a simple taxonomy that divides the approaches proposed in literature into three classes: bot detection systems based on social network information; systems based on crowdsourcing and leveraging human intelligence; and, machine-learning methods based on the identification of highly revealing features that discriminate between bots and humans. Sometimes a hard categorization of a detection strategy into one of these three categories is difficult, since some exhibit mixed elements: we present also a section of methods that combine ideas from these three main approaches.

### Graph-Based Social Bot Detection

The challenge of social bot detection has been framed by various teams in an adversarial setting.<sup>3</sup> One example of this framework is represented by the Facebook Immune System:<sup>30</sup> An adversary may control multiple social bots (often referred to as *sybils* in this context) to impersonate different identities and launch an attack or infiltra-



**The computing community is engaging in the design of advanced methods to automatically detect social bots, or to discriminate between humans and bots.**



tion. Proposed strategies to detect sybil accounts often rely on examining the structure of a social graph. SybilRank,<sup>9</sup> for example, assumes that sybil accounts exhibit a small number of links to legitimate users, instead connecting mostly to other sybils, as they need a large number of social ties to appear trustworthy. This feature is exploited to identify densely interconnected groups of sybils. One common strategy is to adopt off-the-shelf community detection methods to reveal such tightly knit local communities; however, the choice of the community detection algorithm has proven to crucially affect the performance of the detection algorithms.<sup>34</sup> A wise attacker may counterfeit the connectivity of the controlled sybil accounts to mimic the features of the community structure of the portion of the social network populated by legitimate accounts; this strategy would make the attack invisible to methods solely relying on community detection.

To address this shortcoming, some detection systems, for example SybilRank, also employ the paradigm of innocent by association: an account interacting with a legitimate user is considered itself legitimate. Souche<sup>41</sup> and Anti-Reconnaissance<sup>27</sup> also rely on the assumption that social network structure alone separates legitimate users from bots. Unfortunately, the effectiveness of such detection strategies is bound by the behavioral assumption that legitimate users refuse to interact with unknown accounts. This was proven unrealistic by various experiments:<sup>7,16,31</sup> A large-scale social bot infiltration on Facebook showed that over 20% of legitimate users accept friendship requests indiscriminately, and over 60% accept requests from accounts with at least one contact in common.<sup>7</sup> On other platforms like Twitter and Tumblr, connecting and interacting with strangers is one of the main features. In these circumstances, the innocent-by-association paradigm yields high false-negative rates. Some authors noted the limits of the assumption of finding groups of social bots or legitimate users only: real platforms may contain many mixed groups of legitimate users who fell prey of some bots,<sup>3</sup> and sophisticated bots may succeed in large-scale infiltrations making it impossible to detect them

solely from network structure information. This brought Alvisi et al.<sup>3</sup> to recommend a portfolio of complementary detection techniques, and the manual identification of legitimate social network users to aid in the training of supervised learning algorithms.

### Crowdsourcing Social Bot Detection

Wang et al.<sup>38</sup> have explored the possibility of human detection, suggesting the crowdsourcing of social bot detection to legions of workers. As a proof-of-concept, they created an On-line Social Turing Test platform. The authors assumed that bot detection is a simple task for humans, whose ability to evaluate conversational nuances like sarcasm or persuasive language, or to observe emerging patterns and anomalies, is yet unparalleled by machines. Using data from Facebook and Renren (a popular Chinese online social network), the authors tested the efficacy of humans, both expert annotators and workers hired online, at detecting social bot accounts simply from the information on their profiles. The authors observed the detection rate for hired workers drops off over time, although it remains good enough to be used in a majority voting protocol: the same profile is shown to multiple workers and the opinion of the majority determines the final verdict. This strategy exhibits a near-zero false positive rate, a very desirable feature for a service provider.

Three drawbacks undermine the feasibility of this approach: first, although the authors make a general claim that crowdsourcing the detection of social bots might work if implemented since the early stage, this solution might not be cost effective for a platform with a large pre-existing user base, like Facebook and Twitter. Second, to guarantee that a minimal number of human annotators can be employed to minimize costs, “expert” workers are still needed to accurately detect fake accounts, as the “average” worker does not perform well individually. As a result, to reliably build a ground-truth of annotated bots, large social network companies like Facebook and Twitter are forced to hire teams of expert analysts,<sup>30</sup> however such a choice might not be suit-

**Classes of features employed by feature-based systems for social bot detection.**

| Class     | Description                                                                                                                                                                                                                                                                                     |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Network   | Network features capture various dimensions of information diffusion patterns. Statistical features can be extracted from networks based on retweets, mentions, and hashtag co-occurrence. Examples include degree distribution, clustering coefficient, and centrality measures. <sup>29</sup> |
| User      | User features are based on Twitter meta-data related to an account, including language, geographic locations, and account creation time.                                                                                                                                                        |
| Friends   | Friend features include descriptive statistics relative to an account's social contacts, such as median, moments, and entropy of the distributions of their numbers of followers, followees, and posts.                                                                                         |
| Timing    | Timing features capture temporal patterns of content generation (tweets) and consumption (retweets); examples include the signal similarity to a Poisson process, <sup>18</sup> or the average time between two consecutive posts.                                                              |
| Content   | Content features are based on linguistic cues computed through natural language processing, especially part-of-speech tagging; examples include the frequency of verbs, nouns, and adverbs in tweets.                                                                                           |
| Sentiment | Sentiment features are built using general-purpose and Twitter-specific sentiment analysis algorithms, including happiness, arousal-dominance-valence, and emotion scores. <sup>5,19</sup>                                                                                                      |

able for small social networks in their early stages (an issue at odds with the previous point). Finally, exposing personal information to external workers for validation raises privacy issue.<sup>15</sup> While Twitter profiles tend to be more public compared to Facebook, Twitter profiles also contain less information than Facebook or Renren, thus giving a human annotator less ground to make a judgment. Analysis by manual annotators of interactions and content produced by a Syrian social botnet active in Twitter for 35 weeks suggests that some advanced social bots may no longer aim at mimicking human behavior, but rather at misdirecting attention to irrelevant information.<sup>1</sup>

Such smoke screening strategies require high coordination among the bots. This observation is in line with early findings on political campaigns orchestrated by social bots, which exhibited not only peculiar network connectivity patterns but also enhanced levels of coordinated behavior.<sup>28</sup> The idea of leveraging information about the synchronization of account activities has been fueling many social bot detection systems: frameworks like CopyCatch,<sup>4</sup> SynchroTrap,<sup>10</sup> and the Renren Sybil detector<sup>37,42</sup> rely explicitly on the identification of such coordinated behavior to identify social bots.

### Feature-Based Social Bot Detection

The advantage of focusing on behav-

ioral patterns is that these can be easily encoded in features and adopted with machine learning techniques to learn the signature of human-like and bot-like behaviors. This allows for classifying accounts later according to their observed behaviors. Different classes of features are commonly employed to capture orthogonal dimensions of users' behaviors, as summarized in the accompanying table.

One example of a feature-based system is represented by Bot or Not?. Released in 2014, it was the first social bot detection interface for Twitter to be made publicly available to raise awareness about the presence of social bots.<sup>13,e</sup> Similarly to other feature-based systems,<sup>29</sup> Bot or Not? implements a detection algorithm relying upon highly predictive features that capture a variety of suspicious behaviors and well separate social bots from humans. The system employs off-the-shelf supervised learning algorithms trained with examples of both humans and bots behaviors, based on the Texas A&M dataset<sup>24</sup> that contains 15,000 examples of each class and millions of tweets. Bot or Not? scores a detection accuracy above 95%,<sup>f</sup> measured by AU-

e As of the time of this writing, Bot or Not? remains the only social bot detection system with a public-facing interface: <http://truthy.indiana.edu/botornot>

f Detecting more recent and sophisticated social bots, compared to those in the 2011 dataset, may well yield lower accuracy.

**Figure 1. Common features used for social bot detection. (a) The network of hashtags co-occurring in the tweets of a given user. (b) Various sentiment signals including emoticon, happiness and arousal-dominance-valence scores. (c) The volume of content produced and consumed (tweeting and retweeting) over time.**

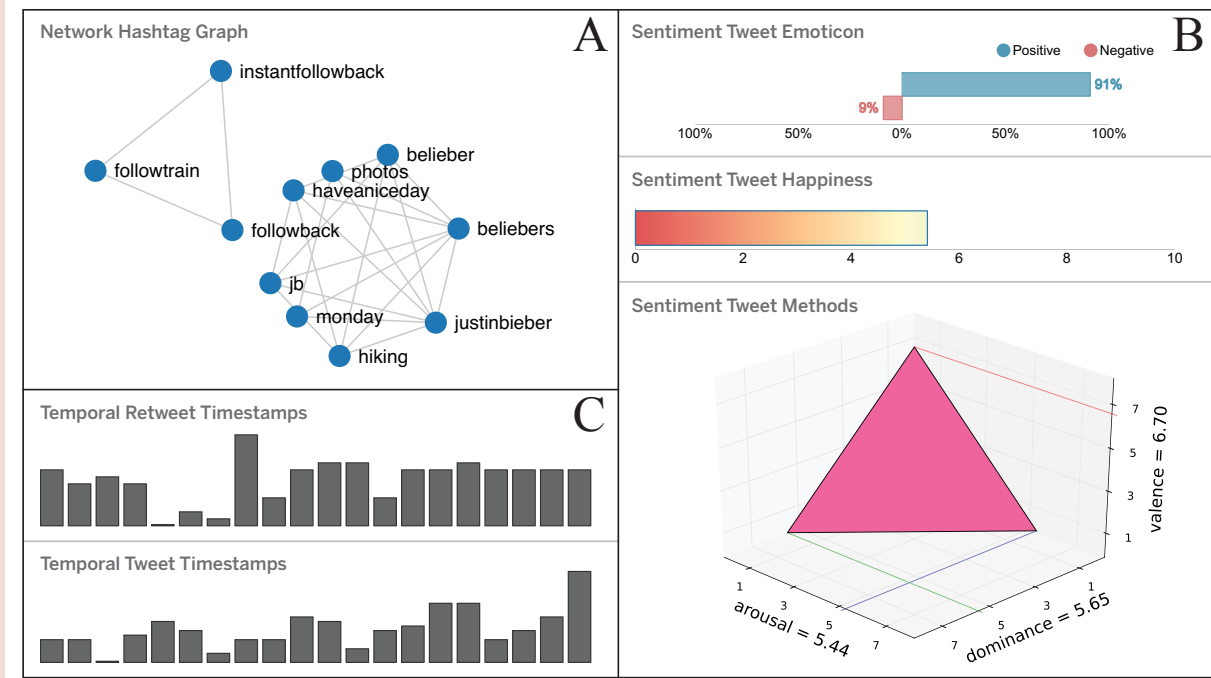
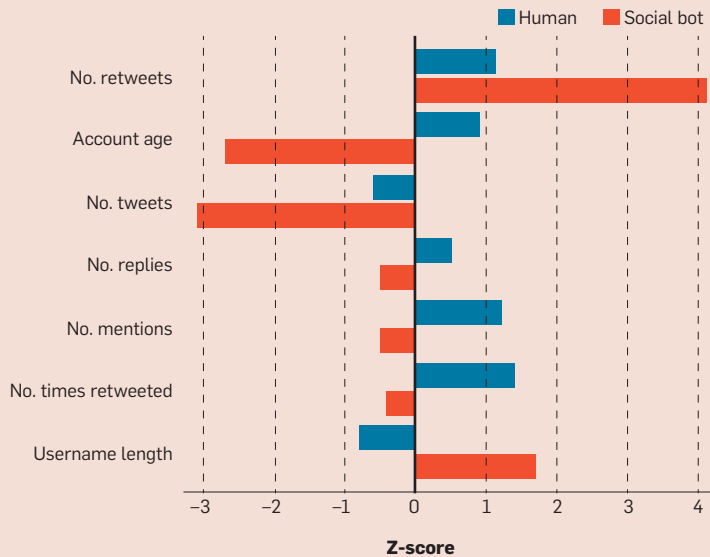


**Figure 2. User behaviors that best discriminate social bots from humans.**

Social bots retweet more than humans and have longer user names, while they produce fewer tweets, replies and mentions, and they are retweeted less than humans. Bot accounts also tend to be more recent.



ROC via cross validation. In addition to the classification results, Bot or Not? features a variety of interactive visualizations that provide insights on the features exploited by the system (see Figure 1 for examples).

Bots are continuously changing and evolving: the analysis of the highly predictive behaviors that feature-based systems can detect may reveal interesting patterns and provide unique opportunities to understand

how to discriminate between bots and humans. User meta-data is considered among the most predictive feature and the most interpretable ones.<sup>22,38</sup> We can suggest a few rules of thumb to infer whether an account is likely a bot, by comparing its meta-data with that of legitimate users (see Figure 2). Further work, however, will be needed to detect sophisticated strategies exhibiting a mixture of humans and social bots features (sometimes referred to as cyborgs). Detecting these bots, or hacked accounts,<sup>43</sup> is currently impossible for feature-based systems.

### Combining Multiple Approaches

Alvisi et al.<sup>3</sup> recognized first the need of adopting complementary detection techniques to effectively deal with sybil attacks in social networks. The Renren Sybil detector<sup>37,42</sup> is an example of system that explores multiple dimensions of users' behaviors like activity and timing information. Examination of ground-truth click-stream data shows that real users spend comparatively more time messaging and looking at other users' contents (such as photos and videos),

whereas Sybil accounts spend their time harvesting profiles and befriending other accounts. Intuitively, social bot activities tend to be simpler in terms of variety of behavior exhibited. By also identifying highly predictive features such as invitation frequency, outgoing requests accepted, and network clustering coefficient, Renren is able to classify accounts into two categories: bot-like and human-like prototypical profiles.<sup>42</sup> Sybil accounts on Renren tend to collude and work together to spread similar content: this additional signal, encoded as content and temporal similarity, is used to detect colluding accounts. In some ways, the Renren approach<sup>37,42</sup> combines the best of network- and behavior-based conceptualizations of Sybil detection. By achieving good results even utilizing only the last 100 click events for each user, the Renren system obviates the need to store and analyze the entire click history for every user. Once the parameters are tweaked against ground truth, the algorithm can be seeded with a fixed number of known legitimate accounts and then used for mostly unsupervised classification. The “Sybil until proven otherwise” approach (the opposite of the innocent-by-association strategy) baked into this framework does lend itself to detecting previously unknown methods of attack: the authors recount the case of spambots embedding text in images to evade detection by content analysis and URL blacklists. Other systems implementing mixed methods, like CopyCatch<sup>4</sup> and SynchroTrap,<sup>10</sup> also score comparatively low false positive rates with respect to, for example, network-based methods.

### Master of Puppets

If social bots are the puppets, additional efforts will have to be directed at finding their “masters.” Governments<sup>g</sup> and other entities with sufficient resources<sup>h</sup> have been alleged to use social bots to their advantage.

**If social bots are the puppets, additional efforts will have to be directed at finding their “masters.”**

Assuming the availability of effective detection technologies, it will be crucial to reverse engineer the observed social bot strategies: who they target, how they generate content, when they take action, and what topics they talk about. A systematic extrapolation of such information may enable identification of the puppet masters.

Efforts in the direction of studying platforms vulnerability have already started. Some researchers,<sup>17</sup> for example, reverse-engineer social bots reporting alarming results: simple automated mechanisms that produce contents and boost followers yield successful infiltration strategies and increase the social influence of the bots. Other teams are creating bots themselves: Tim Hwang’s<sup>22</sup> and Sune Lehmann’s<sup>i</sup> groups continuously challenge our understanding of what strategies effective bots employ, and help quantify the susceptibility of people to their influence.<sup>35,36</sup> Briscoe et al.<sup>8</sup> studied the deceptive cues of language employed by influence bots. Tools like Bot or Not? have been made available to the public to shed light on the presence of social bots online.

Yet many research questions remain open. For example, nobody knows exactly how many social bots populate social media, or what share of content can be attributed to bots—estimates vary wildly and we might have observed only the tip of the iceberg. These are important questions for the research community to pursue, and initiatives such as DARPA’s SMISC bot detection challenge, which took place in the spring of 2015, can be effective catalysts of this emerging area of inquiry.<sup>32</sup>

Bot behaviors are already quite sophisticated: they can build realistic social networks and produce credible content with human-like temporal patterns. As we build better detection systems, we expect an arms race similar to that observed for spam in the past.<sup>21</sup> The need for training instances is an intrinsic limitation of supervised learning in such a scenario; machine learning techniques such as active learning might help respond to newer threats. The race will be over

g Russian Twitter political protests ‘swamped by spam’; [www.bbc.com/news/technology-16108876](http://www.bbc.com/news/technology-16108876)

h Fake Twitter accounts used to promote tar sands pipeline; [www.theguardian.com/environment/2011/aug/05/fake-twitter-tar-sands-pipeline](http://www.theguardian.com/environment/2011/aug/05/fake-twitter-tar-sands-pipeline)

i You are here because of a robot; [sunelehmann.com/2013/12/04/youre-here-because-of-a-robot/](http://sunelehmann.com/2013/12/04/youre-here-because-of-a-robot/)

only when the effectiveness of early detection will sufficiently increase the cost of deception.

The future of social media ecosystems might already point in the direction of environments where machine-machine interaction is the norm, and humans navigate a world populated mostly by bots. We believe there is a need for bots and humans to be able to recognize each other, to avoid bizarre, or even dangerous, situations based on false assumptions of human interlocutors.<sup>j</sup>

### Acknowledgments

The authors are grateful to Qiaozhu Mei, Zhe Zhao, Mohsen JafariAsbagh, Prashant Shiralkar, and Aram Galstyan for helpful discussions.

This work is supported in part by the Office of Naval Research (grant N15A-020-0053), National Science Foundation (grant CCF-1101743), DARPA (grant W911NF-12-1-0037), and the James McDonnell Foundation (grant 220020274). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript. **C**

<sup>j</sup> That Time 2 Bots Were Talking, and Bank of America Butted In; [www.theatlantic.com/technology/](http://www.theatlantic.com/technology/)

### References

- Abokhodair, N., Yoo, D. and McDonald, D.W. Dissecting a social botnet: Growth, content, and influence in Twitter. In *Proceedings of the 18th ACM Conference on Computer-Supported Cooperative Work and Social Computing* (2015). ACM.
- Aiello, L.M., Deplano, M., Schifanella, R. and Ruffo, G. People are strange when you're a stranger: Impact and influence of bots on social networks. In *Proceedings of the 6th AAAI International Conference on Weblogs and Social Media* (2012). AAAI, 10–17.
- Alvisi, L., Clement, A., Epasto, A., Lattanzi, S. and Panconesi, A. Sok: The evolution of sybil defense via social networks. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*. IEEE, 382–396.
- Beutel, A., Xu, W., Guruswami, V., Palow, C. and Faloutsos, C. Copy-Catch: stopping group attacks by spotting lockstep behavior in social networks. In *Proceedings of the 22nd International Conference on World Wide Web* (2013), 119–130.
- Bollen, J., Mao, H. and Zeng, X. Twitter mood predicts the stock market. *J. Computational Science* 2, 1 (2011), 1–8.
- Boshmaf, Y., Muslukhov, I., Beznosov, K. and Ripeanu, M. Key challenges in defending against malicious socialbots. In *Proceedings of the 5th USENIX Conference on Large-scale Exploits and Emergent Threats*, Vol. 12 (2012).
- Boshmaf, Y., Muslukhov, I., Beznosov, K. and Ripeanu, M. 2013. Design and analysis of a social botnet. *Computer Networks* 57, 2 (2013), 556–578.
- Briscoe, E.J., Applin, D.S. and Hayes, H. Cues to deception in social media communications. In *Proceedings of the 47th Hawaii International Conference on System Sciences* (2014). IEEE, 1435–1443.
- Cao, Q., Sirivianos, M., Yang, X. and Pregueiro, T. Aiding the detection of fake accounts in large scale social online services. *NSDI* (2012). 197–210.
- Cao, Q., Yang, X., Yu, J. and Palow, C. Uncovering large groups of active malicious accounts in online social networks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 477–488.
- Cassa, C.A., Chunara, R., Mandl, K. and Brownstein, J.S. Twitter as a sentinel in emergency situations: Lessons from the Boston marathon explosions. *PLoS Currents: Disasters* (July 2013); <http://dx.doi.org/10.1371/currents.dis.ad70cd1c8bc585e9470046cde334ee4b>
- Conover, M., Ratkiewicz, J., Francisco, M., Gonçalves, B., Menczer, F. and Flammini, A. Political polarization on Twitter. In *Proceedings of the 5th International AAAI Conference on Weblogs and Social Media* (2011), 89–96.
- Davis, C.A., Varol, O., Ferrara, E., Flammini, A. and Menczer, F. BotOrNot: A system to evaluate social bots. In *Proceedings of the 25th International World Wide Web Conference Companion* (2016); <http://dx.doi.org/10.1145/2872518.2889302> Forthcoming. Preprint arXiv:1602.00975.
- Edwards, C., Edwards, A., Spence, P.R. and Shelton, A.K. Is that a bot running the social media feed? Testing the differences in perceptions of communication quality for a human agent and a bot agent on Twitter. *Computers in Human Behavior* 33 (2014), 372–376.
- Elovici, Y., Fire, M., Herzberg, A. and Shulman, H. Ethical considerations when employing fake identities in online social networks for research. *Science and Engineering Ethics* (2013), 1–17.
- Elyashar, A., Fire, M., Kagan, D. and Elovici, Y. Homing socialbots: Intrusion on a specific organization's employee using Socialbots. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. ACM, 1358–1365.
- Freitas, C.A. et al. Reverse engineering socialbot infiltration strategies in Twitter. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. ACM 2015.
- Ghosh, R., Surachawala, T. and Lerman, K. Entropy-based classification of "retweeting" activity on Twitter. In *Proceedings of the KDD Workshop on Social Network Analysis* (2011).
- Golder, S.A. and Macy, M.W. Diurnal and seasonal mood vary with work, sleep, and daylength across diverse cultures. *Science* 333, 6051 (2011), 1878–1881.
- Gupta, A., Lamba, H. and Kumaraguru, P. \$1.00 per RT #BostonMarathon #PrayForBoston: Analyzing fake content on Twitter. eCrime Researchers Summit. IEEE (2013), 1–12.
- Heymann, P., Koutrika, G. and Garcia-Molina, H. Fighting spam on social web sites: A survey of approaches and future challenges. *Internet Computing* 11, 6 (2007). IEEE, 36–45.
- Hwang, T., Pearce, I. and Nanis, M. Socialbots: Voices from the fronts. *ACM Interactions* 19, 2 (2012), 38–45.
- Kramer, A.D. Guillory, J.E. and Hancock, J.T. Experimental evidence of massive-scale emotional contagion through social networks. In *Proceedings of the National Academy of Sciences* (2014), 201320040.
- Lee, K., Eoff, B.D., and Caverlee, J. Seven months with the devils: A long-term study of content polluters on Twitter. In *Proceedings of the 5th International AAAI Conference on Weblogs and Social Media* (2011), 185–192.
- Messias, J., Schmidt, L., Oliveira, R. and Benevenuto, F. You followed my bot! Transforming robots into influential users in Twitter. *First Monday* 18, 7 (2013).
- Metaxas, P.T. and Mustafaraj, E. Social media and the elections. *Science* 338, 6106 (2012), 472–473.
- Paradise, A., Puzis, R. and Shabtai, A. Anti-reconnaissance tools: Detecting targeted socialbots. *Internet Computing* 18, 5 (2014), 11–19.
- Ratkiewicz, J., Conover, M., Meiss, M., Gonçalves, B., Flammini, A. and Menczer, F. Detecting and tracking political abuse in social media. In *Proceedings of the 5th International AAAI Conference on Weblogs and Social Media* (2011), 297–304.
- Ratkiewicz, J., Conover, M., Meiss, M., Gonçalves, B., Patil, S., Flammini, A. and Menczer, F. Truthy: Mapping the spread of astroturf in microblog streams. In *Proceedings of the 20th International Conference on the World Wide Web* (2011), 249–252.
- Stein, T., Chen, E. and Mangla, K. Facebook immune system. In *Proceedings of the 4th Workshop on Social Network Systems* (2011). ACM, 8.
- Stringhini, G., Kruegel, C. and Vigna, G. Detecting spammers on social networks. In *Proceedings of the 26th Annual Computer Security Applications Conference* (2010), ACM, 1–9.
- Subrahmanian, V.S., Azaria, A., Durst, S., Kagan, V., Galstyan, A., Lerman, K., Zhu, L., Ferrara, E., Flammini, A., Menczer, F. and others. The DARPA Twitter Bot Challenge. *IEEE Computer* (2016). In press. Preprint arXiv:1601.05140.
- Turing, A.M. Computing machinery and intelligence. *Mind* 49, 236 (1950), 433–460.
- Viswanath, B., Post, A., Gummadi, K.P. and Mislove, A. An analysis of social network-based sybil defenses. *ACM SIGCOMM Computer Communication Review* 41, 4 (2011), 363–374.
- Wagner, C., Mitter, S., Körner, S. and Strohmaier, M. When social bots attack: Modeling susceptibility of users in online social networks. In *Proceedings of the 21st International Conference on World Wide Web* (2012), 41–48.
- Wald, R., Khoshgoftaar, T.M., Napolitano, A. and Summer, C. Predicting susceptibility to social bots on Twitter. In *Proceedings of the 14th IEEE International Conference on Information Reuse and Integration*. IEEE, 6–13.
- Wang, G., Konolige, T., Wilson, C., Wang, X., Zheng, H. and Zhao, B.Y. You are how you click: Clickstream analysis for sybil detection. *USENIX Security* (2013), 241–256.
- Wang, G., Mohanlal, M., Wilson, C., Wang, X., Metzger, M., Zheng, H. and Zhao, B.Y. Social Turing tests: Crowdsourcing sybil detection. NDSS. The Internet Society, 2013.
- Weizenbaum, J. ELIZA—A computer program for the study of natural language communication between man and machine. *Commun. ACM* 9, 1 (Sept. 1966), 36–45.
- Wu, X., Feng, Z., Fan, W., Gao, J. and Yu, Y. Detecting marionette microblog users for improved information credibility. *Machine Learning and Knowledge Discovery in Databases*. Springer, 2013, 483–498.
- Xie, Y., Yu, F., Ke, Q., Abadi, M., Gillum, E., Vitaldevaria, K., Walter, J., Huang, J. and Mao, Z.M. Innocent by association: Early recognition of legitimate users. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. ACM, 353–364.
- Yang, Z., Wilson, C., Wang, X., Gao, T., Zhao, B.Y. and Dai, Y. 2014. Uncovering social network sybils in the wild. *ACM Trans. Knowledge Discovery from Data* 8, 1 (2014), 2.
- Zangerle, E. and Specht, G. 'Sorry, I was hacked' A classification of compromised Twitter accounts. In *Proceedings of the 29th Symposium On Applied Computing* (2014).

**Emilio Ferrara** (emiliofe@usc.edu) is a research assistant professor at the University of Southern California, Los Angeles, and a computer scientist at the USC Information Sciences Institute. He was a postdoctoral fellow at Indiana University when this work was carried out.

**Onur Varol** (ovarol@indiana.edu) is a Ph.D. candidate at Indiana University, Bloomington, IN.

**Clayton Davis** (claydavi@indiana.edu) is a Ph.D. candidate at Indiana University, Bloomington, IN.

**Filippo Menczer** (fil@indiana.edu) is a professor of computer science and informatics at Indiana University, Bloomington, IN.

**Alessandro Flammini** (aflammin@indiana.edu) is an associate professor of informatics at Indiana University, Bloomington, IN.

Copyright held by authors.  
Publication rights licensed to ACM. \$15.00



Watch the authors discuss their work in this exclusive *Communications* video.  
<http://cacm.acm.org/videos/the-rise-of-social-bots>

# research highlights

---

P. 106

## **Technical Perspective Combining Logic and Probability**

By Henry Kautz and Parag Singla

P. 107

## **Probabilistic Theorem Proving**

By Vibhav Gogate and Pedro Domingos

---

P. 116

## **Technical Perspective Mesa Takes Data Warehousing to New Heights**

By Sam Madden

P. 117

## **Mesa: A Geo-Replicated Online Data Warehouse for Google's Advertising System**

By Ashish Gupta, Fan Yang, Jason Govig, Adam Kirsch, Kelvin Chan, Kevin Lai, Shuo Wu, Sandeep Dhoot, Abhilash Rajesh Kumar, Ankur Agiwal, Sanjay Bhansali, Mingsheng Hong, Jamie Cameron, Masood Siddiqi, David Jones, Jeff Shute, Andrey Gubarev, Shivakumar Venkataraman, and Divyakant Agrawal

# Technical Perspective

## Combining Logic and Probability

By Henry Kautz and Parag Singla

A GOAL OF research in artificial intelligence and machine learning since the early days of expert systems has been to develop automated reasoning methods that combine logic and probability. Probabilistic theorem proving (PTP) unifies three areas of research in computer science: reasoning under uncertainty, theorem-proving in first-order logic, and satisfiability testing for propositional logic.

Why is there a need to combine logic and probability? Probability theory allows one to quantify uncertainty over a set of propositions—ground facts about the world—and a probabilistic reasoning system allows one to infer the probability of unknown (hidden) propositions conditioned on the knowledge of other propositions. However, probability theory alone has nothing to say about how propositions are constructed from relationships over entities or tuples of entities, and how general knowledge at the level of relationships is to be represented and applied.

Handling relations takes us into the domain of first-order logic. An important case is collective classification, where the hidden properties of a set of entities depend in part on the relationships between the entities. For example, the probability that a woman contracts cancer is not independent of her sister contracting cancer. Many applications in medicine, biology, natural language processing, computer vision, the social sciences, and other domains require reasoning about relationships under uncertainty.

Researchers in AI have proposed a number of representation languages and algorithms for combining logic and probability over the past decade, culminating in the emergence of the research area named statistical relational learning (SRL).

The initial approaches to SRL used a logical specification of a domain annotated with probabilities (or weights) as a template for instantiating, or grounding, a propositional probabilistic representation, which is then

solved by a traditional probabilistic reasoning engine. More recently, research has centered on the problem of developing algorithms for probabilistic reasoning that can efficiently handle formulas containing quantified variables without grounding—a process called “lifted inference.”

Well before the advent of SRL, work in automated theorem-proving had split into two camps. One pursued algorithms for quantified logic based on the syntactic manipulation of formulas using unification and resolution. The other pursued algorithms for propositional logic based on model finding; that is, searching the space of truth assignments for one that makes the formula true. At the risk of oversimplifying history, the most successful approach for fully automated theorem-proving is model finding by depth-first search over partial truth assignments, the Davis-Putnam-Logemann-Loveland procedure (DPLL).

There is a simple but profound connection between model finding and propositional probabilistic reasoning. Suppose each truth assignment, or model, came with an associated positive weight, and the weights summed to (or could be normalized to sum to) one. This defines a probability distribution. DPLL can be modified in a straightforward manner to either find the most heavily weighted model or to compute the sum of the weights of the

models that satisfy a formula.

The former can be used to perform Maximum a Posteriori (MAP) inference, and the latter, called “weighted model counting,” to perform marginal inference or to compute the partition function.

PTP provides the final step in unifying probabilistic, propositional, and first-order inference. PTP lifts a weighted version of DPLL by allowing it to branch on a logical expression containing uninstantiated variables. In the best case, PTP can perform weighted model counting while only grounding a small part of a large relational probabilistic theory.

SRL is a highly active research area, where many of the ideas in PTP appear in various forms. There are lifted versions of other exact inference algorithms such as variable elimination, as well as lifted versions of approximate algorithms such as belief propagation and variational inference. Approximate inference is often the best one can hope for in large, complex domains. Gogate and Domingos suggest how PTP could be turned in a fast approximate algorithm by sampling from the set of children of a branch point.

PTP sparks many interesting directions for future research. Algorithms must be developed to quickly identify good literals for lifted branching and decomposition. Approximate versions of PTP need to be fleshed out and evaluated against traditional methods for probabilistic inference. Finally, the development of a lifted version of DPLL suggests that researchers working on logical theorem proving revisit the traditional divide between syntactic methods for quantified logics and model-finding methods for propositional logic. 

**There is a simple but profound connection between model finding and propositional probabilistic reasoning.**

**Henry Kautz** is the Robin and Tim Wentworth Director of the Goergen Institute for Data Science and a professor of computer science at the University of Rochester, New York.

**Parag Singla** is an assistant professor of computer science at the Indian Institute of Technology, New Delhi.

Copyright held by authors.

# Probabilistic Theorem Proving

By Vibhav Gogate and Pedro Domingos

## Abstract

Many representation schemes combining first-order logic and probability have been proposed in recent years. Progress in unifying logical and probabilistic inference has been slower. Existing methods are mainly variants of lifted variable elimination and belief propagation, neither of which take logical structure into account. We propose the first method that has the full power of both graphical model inference and first-order theorem proving (in finite domains with Herbrand interpretations). We first define probabilistic theorem proving (PTP), their generalization, as the problem of computing the probability of a logical formula given the probabilities or weights of a set of formulas. We then show how PTP can be reduced to the problem of lifted weighted model counting, and develop an efficient algorithm for the latter. We prove the correctness of this algorithm, investigate its properties, and show how it generalizes previous approaches. Experiments show that it greatly outperforms lifted variable elimination when logical structure is present. Finally, we propose an algorithm for approximate PTP, and show that it is superior to lifted belief propagation.

## 1. INTRODUCTION

Unifying first-order logic and probability enables uncertain reasoning over domains with complex relational structure, and is a long-standing goal of AI. Proposals go back to at least Nilsson,<sup>17</sup> with substantial progress within the community that studies uncertainty in AI starting in the 1990s (e.g., Bacchus,<sup>1</sup> Halpern,<sup>14</sup> Wellman<sup>25</sup>), and added impetus from the new field of statistical relational learning starting in the 2000s.<sup>11</sup> Many well-developed representations now exist (e.g., DeRaedt,<sup>7</sup> and Domingos<sup>10</sup>), but the state of inference is less advanced. For the most part, inference is still carried out by converting models to propositional form (e.g., Bayesian networks) and then applying standard propositional algorithms. This typically incurs an exponential blowup in the time and space cost of inference, and forgoes one of the chief attractions of first-order logic: the ability to perform *lifted* inference, that is, reason over large domains in time independent of the number of objects they contain, using techniques like resolution theorem proving.<sup>20</sup>

In recent years, progress in lifted probabilistic inference has picked up. An algorithm for lifted variable elimination was proposed by Poole<sup>18</sup> and extended by de Salvo Braz<sup>8</sup> and others. Lifted belief propagation was introduced by Singla and Domingos.<sup>24</sup> These algorithms often yield impressive efficiency gains compared to propositionalization, but still fall well short of the capabilities of first-order theorem proving, because they ignore logical structure, treating potentials as black boxes. This paper proposes the first full-blown probabilistic theorem prover that is capable of exploiting both lifting and logical structure, and includes standard theorem proving and graphical model inference as special cases.

Our solution is obtained by reducing probabilistic theorem proving (PTP) to lifted weighted model counting. We first do the corresponding reduction for the propositional case, extending previous work by Sang et al.<sup>22</sup> and Chavira and Darwiche.<sup>3</sup> We then lift this approach to the first-order level, and refine it in several ways. We show that our algorithm can be exponentially more efficient than first-order variable elimination, and is never less efficient (up to constants). For domains where exact inference is not feasible, we propose a sampling-based approximate version of our algorithm. Finally, we report experiments in which PTP greatly outperforms first-order variable elimination and belief propagation, and discuss future research directions.

## 2. LOGIC AND THEOREM PROVING

We begin with a brief review of propositional logic, first-order logic and theorem proving.

The simplest formulas in propositional logic are *atoms*: individual symbols representing propositions that may be true or false in a given world. More complex formulas are recursively built up from atoms and the logical connectives  $\neg$  (negation),  $\wedge$  (conjunction),  $\vee$  (disjunction),  $\Rightarrow$  (implication) and  $\Leftrightarrow$  (equivalence). For example,  $\neg A \vee (B \wedge C)$  is true iff  $A$  is false or  $B$  and  $C$  are true. A *knowledge base* ( $KB$ ) is a set of logical formulas. The fundamental problem in logic is determining *entailment*, and algorithms that do this are called *theorem provers*. A knowledge base  $K$  entails a query formula  $Q$  iff  $Q$  is true in all worlds in which all formulas in  $K$  are true, a *world* being an assignment of truth values to all atoms.

A world is a *model* of a  $KB$  iff the  $KB$  is true in it. Theorem provers typically first convert  $K$  and  $Q$  to *conjunctive normal form* ( $CNF$ ). A  $CNF$  formula is a conjunction of *clauses*, each of which is a disjunction of *literals*, each of which is an atom or its negation. For example, the  $CNF$  of  $\neg A \vee (B \wedge C)$  is  $(\neg A \vee B) \wedge (\neg A \vee C)$ . A *unit clause* consists of a single literal. Entailment can then be computed by adding  $\neg Q$  to  $K$  and determining whether the resulting  $KB$   $K_Q$  is *satisfiable*, that is, whether there exists a world where all clauses in  $K_Q$  are true. If not,  $K_Q$  is unsatisfiable, and  $K$  entails  $Q$ . Algorithm 1 shows this basic theorem proving schema.  $CNF(K)$  converts  $K$  to  $CNF$ , and  $SAT(C)$  returns True if  $C$  is satisfiable and False otherwise.

### Algorithm 1 TP( $KB$ $K$ , query $Q$ )

```

 $K_Q \leftarrow K \cup \{\neg Q\}$ 
return  $\neg SAT(CNF(K_Q))$ 

```

The original version of this paper was published in the *Proceedings of the 27<sup>th</sup> Conference on Uncertainty in Artificial Intelligence*, 2011, AUAI Press, 256–265.

The earliest theorem prover is the Davis–Putnam algorithm (henceforth called DP).<sup>6</sup> It makes use of the *resolution* rule: if a KB contains the clauses  $A_1 \vee \dots \vee A_n$  and  $B_1 \vee \dots \vee B_m$ , where the  $A$ 's and  $B$ 's represent literals, and some literal  $A_i$  is the negation of some literal  $B_j$ , then the clause  $A_1 \vee \dots \vee A_{i-1} \vee A_{i+1} \vee \dots \vee A_n \vee B_1 \vee \dots \vee B_{j-1} \vee B_{j+1} \vee \dots \vee B_m$  can be added to the KB. For each atom  $A$  in the KB, DP resolves every pair of clauses  $C_1, C_2$  in the KB such that  $C_1$  contains  $A$  and  $C_2$  contains  $\neg A$ , and adds the result to the KB. It then deletes all clauses that contain a literal of  $A$  from the KB. If at some point the empty clause is derived, the KB is unsatisfiable, and the query formula (previously negated and added to the KB) is therefore proven to be entailed by the KB. DP is in fact just the variable elimination algorithm for the special case of 0-1 potentials.

Modern propositional theorem provers use the DPLL algorithm,<sup>5</sup> a variant of DP that replaces the elimination step with a *splitting* step: instead of eliminating all clauses containing the chosen atom  $A$ , resolve all clauses in the KB with  $A$ , simplify and recurse, and do the same with  $\neg A$ . If both recursions fail, the KB is unsatisfiable. DPLL has linear space complexity, compared to exponential for Davis–Putnam, and is the basis of the algorithms in this paper.

First-order logic inherits all the features of propositional logic, and in addition allows atoms to have internal structure. An atom is now a predicate symbol, representing a relation in the domain of interest, followed by a parenthesized list of variables and/or constants, representing objects. For example, `Friends(Anna, x)` is an atom. A *ground atom* has only constants as arguments. First-order logic has two additional connectives,  $\forall$  (universal quantification) and  $\exists$  (existential quantification). For example,  $\forall x \text{ Friends(Anna, } x)$  means that Anna is friends with everyone, and  $\exists x \text{ Friends(Anna, } x)$  means that Anna has at least one friend. In this paper, we assume that domains are finite (and therefore function-free) and that there is a one-to-one mapping between constants and objects in the domain (Herbrand interpretations).

As long as the domain is finite, first-order theorem proving can be carried out by *propositionalization*: creating atoms from all possible combinations of predicates and constants, and applying a propositional theorem prover. However, this is potentially very inefficient. A more sophisticated alternative is first-order resolution,<sup>20</sup> which proceeds by resolving pairs of clauses and adding the result to the KB until the empty clause is derived. Two first-order clauses can be resolved if they contain complementary literals that *unify*, that is, there is a *substitution* of variables by constants or other variables that makes the two literals identical up to the negation sign. Conversion to CNF is carried out as before, with the additional step of removing all existential quantifiers by a process called *skolemization*.

First-order logic allows knowledge to be expressed more concisely than propositional logic. For example, the rules of chess can be stated in a few pages in first-order logic, but require hundreds of thousands in propositional logic. Probabilistic logical languages extend this power to uncertain domains. The goal of this paper is to similarly extend the power of first-order theorem proving.

### 3. PROBLEM DEFINITION

Following Nilsson,<sup>17</sup> we define PTP as the problem of determining the probability of an arbitrary query formula  $Q$  given a set of logical formulas  $F_i$  and their probabilities  $P(F_i)$ . For the problem to be well defined, the probabilities must be consistent, and Nilsson<sup>17</sup> provides a method for verifying consistency. Probabilities estimated by maximum likelihood from an observed world are guaranteed to be consistent. In general, a set of formula probabilities does not specify a complete joint distribution over the atoms appearing in them, but one can be obtained by making the *maximum entropy* assumption: the distribution contains no information beyond that specified by the formula probabilities.<sup>17</sup> Finding the maximum entropy distribution given a set of formula probabilities is equivalent to learning a maximum-likelihood log-linear model whose features are the formulas; many algorithms for this purpose are available (iterative scaling, gradient descent, etc.).

We call a set of formulas and their probabilities together with the maximum entropy assumption a *probabilistic knowledge base (PKB)*. Equivalently, a PKB can be directly defined as a log-linear model with the formulas as features and the corresponding weights or potential values. Potentials are the most convenient form, since they allow determinism (0-1 probabilities) without recourse to infinity. If  $\mathbf{x}$  is a world and  $\Phi_i(\mathbf{x})$  is the potential corresponding to formula  $F_i$ , by convention (and without loss of generality) we let  $\Phi_i(\mathbf{x}) = 1$  if  $F_i$  is true, and  $\Phi_i(\mathbf{x}) = \phi_i \geq 0$  if the formula is false. *Hard* formulas have  $\phi_i = 0$  and *soft* formulas have  $\phi_i > 0$ . In order to compactly subsume standard probabilistic models, we interpret a universally quantified formula as a set of features, one for each grounding of the formula, as in Markov logic.<sup>10</sup> A PKB  $\{(F_i, \phi_i)\}$  thus represents the joint distribution

$$P(\mathbf{X}=\mathbf{x}) = \frac{1}{Z} \prod_i \phi_i^{n_i(\mathbf{x})} \quad (1)$$

where  $n_i(\mathbf{x})$  is the number of false groundings of  $F_i$  in  $\mathbf{x}$ , and  $Z$  is a normalization constant (the *partition function*). We can now define PTP succinctly as follows:

---

#### Probabilistic theorem proving (PTP)

*Input:* Probabilistic KB  $K$  and query formula  $Q$

*Output:*  $P(Q|K)$

---

If all formulas are hard, a PKB reduces to a standard logical KB. Determining whether a KB  $K$  logically entails a query  $Q$  is equivalent to determining whether  $P(Q|K) = 1$ .<sup>10</sup> Graphical models can be easily converted into equivalent PKBs.<sup>3</sup> Conditioning on evidence is done by adding the corresponding hard ground atoms to the PKB, and the conditional marginal of an atom is computed by issuing the atom as the query. Thus PTP has both logical theorem proving and inference in graphical models as special cases.

In this paper, we solve PTP by reducing it to lifted weighted model counting. *Model counting* is the problem of determining the number of worlds that satisfy a KB. *Weighted* model counting can be defined as follows.<sup>3</sup> Assign a weight to each literal, and let the weight of a world be the product of the weights of the literals that are true in it. Then weighted

model counting is the problem of determining the sum of the weights of the worlds that satisfy a KB:

#### Weighted model counting (WMC)

*Input:* CNF  $C$  and set of literal weights  $W$

*Output:* Sum of weights of worlds that satisfy  $C$

Figure 1 depicts graphically the set of inference problems addressed by this paper. Generality increases in the direction of the arrows. We first propose an algorithm for propositional weighted model counting and then lift it to the first-order level. The resulting algorithm is applicable to all the problems in the diagram. (Weighted SAT/MPE inference requires replacing sums with maxes with an additional traceback or backtracking step, but we do not pursue it here and leave it for future work.)

#### 4. PROPOSITIONAL CASE

This section generalizes the Bayesian network inference techniques in Darwiche<sup>4</sup> and Sang et al.<sup>22</sup> to arbitrary propositional PKBs, evidence, and query formulas. Although this is of value in its own right, its main purpose is to lay the groundwork for the first-order case.

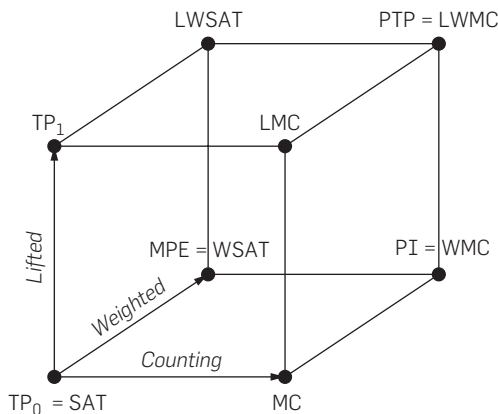
The probability of a formula is the sum of the probabilities of the worlds that satisfy it. Thus to compute the probability of a formula  $Q$  given a PKB  $K$  it suffices to compute the partition function of  $K$  with and without  $Q$  added as a hard formula:

$$P(Q|K) = \frac{\sum_{\mathbf{x}} 1_Q(\mathbf{x}) \prod_i \Phi_i(\mathbf{x})}{Z(K)} = \frac{Z(K \cup \{(Q, 0)\})}{Z(K)} \quad (2)$$

where  $1_Q(\mathbf{x})$  is the indicator function (1 if  $Q$  is true in  $\mathbf{x}$  and 0 otherwise).

In turn, the computation of partition functions can be reduced to weighted model counting using the procedure in Algorithm 2. This replaces each soft formula  $F_i$  in  $K$  by a corresponding hard formula  $F_i \Leftrightarrow A_i$ , where  $A_i$  is a new atom, and assigns to every  $\neg A_i$  literal a weight of  $\phi_i$  (the value of the potential  $\Phi_i$  when  $F_i$  is false).

**Figure 1. Inference problems addressed in this paper.**  $TP_0$  and  $TP_1$  is propositional and first-order theorem proving respectively,  $PI$  is probabilistic inference (computing marginals),  $MPE$  is computing the most probable explanation,  $SAT$  is satisfiability,  $MC$  is model counting,  $W$  is weighted and  $L$  is lifted.  $A = B$  means  $A$  can be reduced to  $B$ .



**THEOREM 1.**  $Z(K) = WMC(WCNF(K))$ .

**PROOF.** If a world violates any of the hard clauses in  $K$  or any of the  $F_i \Leftrightarrow A_i$  equivalences, it does not satisfy  $C$  and is therefore not counted. The weight of each remaining world  $\mathbf{x}$  is the product of the weights of the literals that are true in  $\mathbf{x}$ . By the  $F_i \Leftrightarrow A_i$  equivalences and the weights assigned by  $WCNF(K)$ , this is  $\prod_i \Phi_i(\mathbf{x})$ . (Recall that  $\Phi_i(\mathbf{x}) = 1$  if  $F_i$  is true in  $\mathbf{x}$  and  $\Phi_i(\mathbf{x}) = \phi_i$  otherwise.) Thus  $\mathbf{x}$ 's weight is the unnormalized probability of  $\mathbf{x}$  under  $K$ . Summing these yields the partition function  $Z(K)$ .  $\square$

Equation 2 and Theorem 1 lead to Algorithm 3 for PTP. (Compare with Algorithm 1.)  $WMC(C, W)$  can be any weighted model counting algorithm.<sup>3</sup> Most model counters are variations of Relsat, itself an extension of DPLL.<sup>2</sup> Relsat splits on atoms until the CNF is decomposed into sub-CNFs sharing no atoms, and recurses on each sub-CNF. This decomposition is crucial to the efficiency of the algorithm. In this paper we use a weighted version of Relsat, shown in Algorithm 4.  $A(C)$  is the set of atoms that appear in  $C$ .  $C|A$  denotes the CNF obtained by removing the literal  $A$  and its negation  $\neg A$  from all clauses in which they appear in and setting to *Satisfied* all clauses in which  $A$  appears in. Notice that, unlike in DPLL, satisfied clauses cannot simply be deleted, because we need to keep track of which atoms they are over for model counting purposes. However, they can be ignored in the decomposition step, since they introduce no dependencies. The atom to split on in the splitting step can be chosen using various heuristics.<sup>23</sup>

#### Algorithm 2 $WCNF(PKB K)$

```

for all  $(F_i, \phi_i) \in K$  s.t.  $\phi_i > 0$  do
     $K \leftarrow K \cup \{(F_i \Leftrightarrow A_i, 0)\} \setminus \{(F_i, \phi_i)\}$ 
 $C \leftarrow CNF(K)$ 
for all  $\neg A_i$  literals do  $W_{\neg A_i} \leftarrow \phi_i$ 
for all other literals  $L$  do  $W_L \leftarrow 1$ 
return  $(C, W)$ 

```

#### Algorithm 3 PTP(PKB $K$ , query $Q$ )

```

 $K_Q \leftarrow K \cup \{(Q, 0)\}$ 
return  $WMC(WCNF(K_Q)) / WMC(WCNF(K))$ 

```

#### Algorithm 4 $WMC(CNF C, weights W)$

```

// Base case
if all clauses in  $C$  are satisfied then
    return  $\prod_{A \in A(C)} (W_A + W_{\neg A})$ 
if  $C$  has an empty unsatisfied clause then return 0
// Decomposition step
if  $C$  can be partitioned into CNFs  $C_1, \dots, C_k$  sharing no atoms then
    return  $\prod_{i=1}^k WMC(C_i, W)$ 
// Splitting step
Choose an atom  $A$ 
return  $W_A WMC(C|A; W) + W_{\neg A} WMC(C|\neg A; W)$ 

```

**THEOREM 2.** *Algorithm WMC( $C, W$ ) correctly computes the weighted model count of CNF  $C$  under literal weights  $W$ .*

**PROOF SKETCH.** If all clauses in  $C$  are satisfied, all assignments to the atoms in  $C$  satisfy it, and the corresponding total weight is  $\prod_{A \in \mathcal{A}(C)} (W_A + W_{\neg A})$ . If  $C$  has an empty unsatisfied clause, it is unsatisfiable given the truth assignment so far, and the corresponding weighted count is 0. If two CNFs share no atoms, the WMC of their conjunction is the product of the WMCs of the individual CNFs. Splitting on an atom produces two disjoint sets of worlds, and the total WMC is therefore the sum of the WMCs of the two sets, weighted by the corresponding literal's weight.  $\square$

## 5. FIRST-ORDER CASE

We now lift PTP to the first-order level. We consider first the case of PKBs without existential quantifiers. Algorithms 2 and 3 remain essentially unchanged, except that formulas, literals and CNF conversion are now first-order. In particular, for Theorem 1 to remain true, each new atom  $A_i$  in Algorithm 2 must now consist of a new predicate symbol followed by a parenthesized list of the variables and constants in the corresponding formula  $F_i$ . The proof of the first-order version of the theorem then follows by propositionalization. Lifting Algorithm 4 is the focus of the rest of this section.

We begin with some necessary definitions. A *substitution constraint* is an expression of the form  $x = y$  or  $x \neq y$ , where  $x$  is a variable and  $y$  is either a variable or a constant. (Much richer substitution constraint languages are possible, but we adopt the simplest one that allows PTP to subsume both standard function-free theorem proving and first-order variable elimination.) Two literals are *unifiable* under a set of substitution constraints  $S$  if there exists at least one ground literal consistent with  $S$  that is an instance of both, up to the negation sign. A  $(C, S)$  pair, where  $C$  is a first-order CNF whose variables have been standardized apart and  $S$  is a set of substitution constraints, represents the ground CNF obtained by replacing each clause in  $C$  with the conjunction of its groundings that are consistent with the constraints in  $S$ . For example, using upper case for constants and lower case for variables, and assuming that the PKB contains only two constants  $A$  and  $B$ , if  $C = R(A, B) \wedge (\neg R(x, y) \vee S(y, z))$  and  $S = \{x = y, z \neq A\}$ ,  $(C, S)$  represents the ground CNF  $R(A, B) \wedge (\neg R(A, A) \vee S(A, B)) \wedge (\neg R(B, B) \vee S(B, B))$ . Clauses with equality substitution constraints can be abbreviated in the obvious way (e.g.,  $T(x, y, z)$  with  $x = y$  and  $z = C$  can be abbreviated as  $T(x, x, C)$ ).

We lift the base case, decomposition step, and splitting step of Algorithm 4 in turn. The result is shown in Algorithm 5. In addition to the first-order CNF  $C$  and weights on first-order literals  $W$ , LWMC takes as an argument an initially empty set of substitution constraints  $S$  which, similar to logical theorem proving, is extended along each branch of the inference as the algorithm progresses.

### 5.1. Lifting the base case

The base case changes only by raising each first-order atom  $A$ 's sum of weights to  $n_A(S)$ , the number of groundings of  $A$  compatible with the constraints in  $S$ . This is necessary and

### Algorithm 5 LWMC(CNF $C$ , substs. $S$ , weights $W$ )

// *Lifted base case*

**if** all clauses in  $C$  are satisfied **then**

**return**  $\prod_{A \in \mathcal{A}(C)} (W_A + W_{\neg A})^{n_A(S)}$

**if**  $C$  has an empty unsatisfied clause **then return** 0

// *Lifted decomposition step*

**if** there exists a lifted decomposition  $\{C_{1,1}, \dots, C_{1,m_1}, \dots, C_{k,1}, \dots, C_{k,m_k}\}$  of  $C$  under  $S$  **then**

**return**  $\prod_{i=1}^k [\text{LWMC}(C_{i,1}, S, W)]^{m_i}$

// *Lifted splitting step*

Choose an atom  $A$

Let  $\{\sum_{A,S}^{(1)}, \dots, \sum_{A,S}^{(l)}\}$  be a lifted split of  $A$  for  $C$  under  $S$

**return**  $\sum_{i=1}^l n_i W_A^{t_i} W_{\neg A}^{f_i} \text{LWMC}(C|\sigma_j; S_j, W)$

where  $n_i, t_i, f_i, \sigma_j$  and  $S_j$  are as in Proposition 3

sufficient since each atom  $A$  has  $n_A(S)$  groundings, and all ground atoms are independent because the CNF is satisfied irrespective of their truth values. Note that  $n_A(S)$  is the number of groundings of  $A$  consistent with  $S$  that can be formed using all the constants in the original CNF.

### 5.2. Lifting the decomposition step

Clearly, if  $C$  can be decomposed into two or more CNFs such that no two CNFs share any unifiable literals, a lifted decomposition of  $C$  is possible (i.e., a decomposition of  $C$  into first-order CNFs on which LWMC can be called recursively). But the symmetry of the first-order representation can be further exploited. For example, if the CNF  $C$  can be decomposed into  $k$  CNFs  $C_1, \dots, C_k$  sharing no unifiable literals and such that for all  $i, j$ ,  $C_i$  is identical to  $C_j$  up to a renaming of the variables and constants, then  $\text{LWMC}(C) = [\text{LWMC}(C_1)]^k$ . We formalize these conditions below.

**DEFINITION 1.** *The set of first-order CNFs  $\{C_{1,1}, \dots, C_{1,m_1}, \dots, C_{k,1}, \dots, C_{k,m_k}\}$  is called a lifted decomposition of CNF  $C$  under substitution constraints  $S$  if, given  $S$ , it satisfies the following three properties: (i)  $C = C_{1,1} \wedge \dots \wedge C_{k,m_k}$ ; (ii) no two  $C_{i,j}$ 's share any unifiable literals; (iii) for all  $i, j, j'$ , such that  $j \neq j'$ ,  $C_{i,j}$  is identical to  $C_{i,j'}$ .<sup>a</sup>*

**PROPOSITION 1.** *If  $\{C_{1,1}, \dots, C_{1,m_1}, \dots, C_{k,1}, \dots, C_{k,m_k}\}$  is a lifted decomposition of  $C$  under  $S$ , then*

$$\text{LWMC}(C, S, W) = \prod_{i=1}^k [\text{LWMC}(C_{i,1}, S, W)]^{m_i}. \quad (3)$$

Rules for identifying lifted decompositions can be derived in a straightforward manner from the inversion argument in de Salvo Braz<sup>8</sup> and the power rule in Jha et al.<sup>15</sup> An example of such a rule is given in the definition and proposition below.

**DEFINITION 2.** *A set of variables  $X = \{x_1, \dots, x_m\}$  is called a decomposer of a CNF  $C$  if it satisfies the following three properties: (i) for each clause  $C_j$  in  $C$ , there is exactly one variable  $x_i$  in  $X$  such that  $x_i$  appears in all atoms in  $C_j$ ; (ii) if  $x_i \in X$  appears as an argument of predicate  $R$  (say at position  $k$  in an atom having predicate symbol  $R$ ), then all variables in all clauses that*

<sup>a</sup> Throughout this paper, when we say that two clauses are identical, we mean that they are identical up to a renaming of constants and variables.

appear as the same argument (namely at position  $k$ ) of  $R$  are included in  $X$ ; and (iii) no pair of variables  $x_i, x_j \in X$  such that  $i \neq j$  appear as different arguments of a predicate  $R$  in  $C$ .

For example,  $\{x_1, x_2\}$  is a decomposer of the CNF  $(R(x_1) \vee S(x_1, x_3)) \wedge (R(x_2) \vee T(x_2, x_4))$  while the CNF  $(R(x_1, x_2) \vee S(x_2, x_3)) \wedge (R(x_4, x_5) \vee T(x_4, x_6))$  has no decomposer. Given a decomposer  $\{x_1, \dots, x_m\}$  and a CNF  $C$ , it is easy to see that for every pair of substitutions of the form  $S_x = \{x_1 = X, \dots, x_m = X\}$  and  $S_y = \{x_1 = Y, \dots, x_m = Y\}$ , with  $X \neq Y$ , the CNFs  $C_x$  and  $C_y$  obtained by applying  $S_x$  and  $S_y$  to  $C$  do not share any unifiable literals. A decomposer thus yields a lifted decomposition. Given a CNF, a decomposer can be found in linear time.

When there are no substitution constraints on the variables in the decomposer, as in the example above, all CNFs formed by substituting the variables in the decomposer with the same constant are identical. Thus,  $k = 1$  in Equation (3) and  $m_1$  equals the number of constants (objects) in the PKB. However, when there are substitution constraints, the CNFs may not be identical. For example, given the CNF  $(R(x_1) \vee S(x_1, x_3)) \wedge (R(x_2) \vee T(x_2, x_4))$  and substitution constraints  $\{x_1 \neq A, x_2 \neq B\}$ , the CNF formed by substituting  $\{x_1 = A, x_2 = A\}$  is not identical to the CNF formed by substituting  $\{x_1 = C, x_2 = C\}$ . Specifically, the first CNF is  $(R(A) \vee T(A, x_4))$  (since the clause  $(R(x_1) \vee S(x_1, x_3))$  has no valid groundings for the substitution  $x_1 = A$  given the constraint  $x_1 \neq A$ ) while the second CNF is  $(R(C) \vee S(C, x_3)) \wedge (R(C) \vee T(C, x_4))$ .

A possible approach for solving this problem is illustrated below. For simplicity, assume that each variable  $x$  in the decomposer is involved in exactly one substitution constraint of the form  $x \neq X$  (or  $x = X$ ) where  $X$  is a constant. Consider all possible combinations (Cartesian product) of the constraints and their negation on the decomposer. Observe that for each clause in the CNF, the subset of constants  $O$  that satisfy all constraints in a given combination also satisfy the following property: for any two distinct constants  $X_i$  and  $X_j$  in  $O$ , the clause (possibly having no valid groundings) obtained by substituting the decomposer variable in it by  $X_i$  is identical to the one obtained by substituting the decomposer variable by  $X_j$  (up to a renaming of constants and variables). Thus, a simple approach to decompose the CNF into subsets of identical but disjoint CNFs is to partition the constants, with each part corresponding to a possible combination of the constraints and their negation.

For instance, in our example CNF, given the decomposer  $X = \{x_1, x_2\}$ , and the constraints  $\{x \neq A, x \neq B\}$  where  $x \in X$ , we have the following four combinations of constraints and their negation: (1)  $(x \neq A, x \neq B)$ ; (2)  $(x \neq A, x = B)$ ; (3)  $(x = A, x \neq B)$ ; and (4)  $(x = A, x = B)$ . Notice that the last combination is inconsistent (has no solution) and therefore we can ignore it. Assuming that there are five constants  $\{A, B, C, D, E\}$  in the domain, the three consistent combinations given above yield the following partition of the constants:  $\{\{C, D, E\}, \{A\}, \{B\}\}$ . The three corresponding parts of the lifted decomposition of the CNF are (for readability, we do not standardize variables apart): (1)  $(R(x_1) \vee S(x_1, x_3)) \wedge (R(x_2) \vee T(x_2, x_4)), \{x_1, x_2 \in \{C, D, E\}, x_1 \neq A, x_2 \neq B\}$ ; (2)  $(R(x_1) \vee S(x_1, x_3)) \wedge (R(x_2) \vee T(x_2, x_4)), \{x_1, x_2 \in \{B\}, x_1 \neq A, x_2 \neq B\}$ ; and (3)

$(R(x_1) \vee S(x_1, x_3)) \wedge (R(x_2) \vee T(x_2, x_4)), \{x_1, x_2 \in \{A\}, x_1 \neq A, x_2 \neq B\}$ .

In general, the partitioning problem described above can be solved using constraint satisfaction techniques. In summary:

**PROPOSITION 2.** *Let  $X$  be a decomposer of a CNF  $C$  and let  $S$  be a set of substitution constraints over  $C$ . Let  $\{\{X_{1,1}, \dots, X_{1,m}\}, \dots, \{X_{k,1}, \dots, X_{k,m_k}\}\}$  be a partition of the constants in the domain and let  $C' = \{C_{1,1}, \dots, C_{1,m_1}, \dots, C_{k,1}, \dots, C_{k,m_k}\}$  be such that (i) for all  $i, j, j', j \neq j', C_{i,j}$  is identical to  $C_{i,j'}$  given  $S$ , and (ii)  $C_{i,m_i}$  is a CNF formed by substituting each variable in  $X$  by  $X_{i,m_i}$ . Then  $C'$  is a lifted decomposition of  $C$  under  $S$ .*

### 5.3. Lifting the splitting step

Splitting on a non-ground atom means splitting on all groundings of it consistent with the current substitution constraints  $S$ . Naively, if the atom has  $c$  groundings consistent with  $S$  this will lead to a sum of  $2^c$  recursive calls to LWMC, one for each possible truth assignment to the  $c$  ground atoms. However, in general these calls will have repeated structure and can be replaced by a much smaller number. The lifted splitting step exploits this.

We introduce some notation and definitions. Let  $\sigma_{A,S}$  denote a truth assignment to the groundings of atom  $A$  that is consistent with the substitution constraints  $S$ , and let  $\Sigma_{A,S}$  denote the set of all possible such assignments. Let  $C|_{\sigma_{A,S}}$  denote the CNF formed by removing  $A$  and  $\neg A$  from all clauses that satisfy  $S$ , and setting to *Satisfied* all ground clauses that are satisfied because of  $\sigma_{A,S}$ . This can be done in a lifted manner by updating the substitution constraints associated with each clause. For instance, consider the clause  $R(x) \vee S(x, y)$  and substitution constraint  $\{x \neq A\}$ , and suppose the domain is  $\{A, B, C\}$  (i.e., these are all the constants appearing in the PKB). Given the assignment  $R(B) = \text{True}, R(C) = \text{False}$  and ignoring satisfied clauses, the clause becomes  $S(x, y)$  and the constraint set becomes  $\{x \neq A, x \neq B\}$ .  $R(x)$  is removed from the clause because all of its groundings are instantiated. The constraint  $x \neq B$  is added because the assignment  $R(B) = \text{True}$  satisfies all groundings in which  $x = B$ .

**DEFINITION 3.** *The partition  $\{\Sigma_{A,S}^{(1)}, \dots, \Sigma_{A,S}^{(l)}\}$  of  $\Sigma_{A,S}$  is called a lifted split of atom  $A$  for CNF  $C$  under substitution constraints  $S$  if every part  $\Sigma_{A,S}^{(i)}$  satisfies the following two properties: (i) all truth assignments in  $\Sigma_{A,S}^{(i)}$  have the same number of true atoms; (ii) for all pairs  $\sigma_j, \sigma_k \in \Sigma_{A,S}^{(i)}, C|_{\sigma_j}$  is identical to  $C|_{\sigma_k}$ .*

**PROPOSITION 3.** *If  $\{\Sigma_{A,S}^{(1)}, \dots, \Sigma_{A,S}^{(l)}\}$  is a lifted split of  $A$  for  $C$  under  $S$ , then*

$$LWMC(C, S, W) = \sum_{i=1}^l n_i W_A^{t_i} W_{\neg A}^{f_i} LWMC(C|_{\sigma_i}, S_i, W)$$

where  $n_i = |\Sigma_{A,S}^{(i)}|$ ,  $\sigma \in \Sigma_{A,S}^{(i)}$ ,  $t_i$  and  $f_i$  are the number of true and false atoms in  $\sigma$  respectively, and  $S_i$  is  $S$  augmented with the substitution constraints required to form  $C|_{\sigma}$ .

Again, we can derive rules for identifying a lifted split by using the counting arguments in de Salvo Braz<sup>8</sup> and the generalized binomial rule in Jha et al.<sup>15</sup> We omit the details for lack of space. In the worst case, lifted splitting defaults to splitting on a ground atom. In most inference problems,

the PKB will contain many hard ground unit clauses (the evidence). Splitting on the corresponding ground atoms then reduces to a single recursive call to LWMC for each atom. In general, the atom to split on in Algorithm 5 should be chosen with the goal of yielding lifted decompositions in the recursive calls (e.g., using lifted versions of the propositional heuristics<sup>23</sup>).

Notice that the lifting schemes used for decomposition and splitting in Algorithm 5 by no means exhaust the space of possible probabilistic lifting rules. For example, Jha et al.<sup>15</sup> and Milch et al.<sup>16</sup> contain examples of other lifting rules. Searching for new probabilistic lifted inference rules, and positive and negative theoretical results about what can be lifted, looks like a fertile area for future research.

The theorem below follows from Theorem 2 and the arguments above.

**THEOREM 3.** *Algorithm LWMC( $C, \emptyset, W$ ) correctly computes the weighted model count of CNF  $C$  under literal weights  $W$ .*

#### 5.4. Extensions and discussion

Although most probabilistic logical languages do not include existential quantification, handling it in PTP is desirable for the sake of logical completeness. This is complicated by the fact that skolemization is not sound for model counting (skolemization will not change satisfiability but can change the model count), and so cannot be applied. The result of conversion to CNF is now a conjunction of clauses with universally and/or existentially quantified variables (e.g.,  $[\forall x \exists y (R(x, y) \vee S(y))] \wedge [\exists u \forall v \forall w T(u, v, w)]$ ). Algorithm 5 now needs to be able to handle clauses of this form. If no universal quantifier appears nested inside an existential one, this is straightforward, since in this case an existentially quantified clause is just a compact representation of a longer one. For example, if the domain is  $\{A, B, C\}$ , the unit clause  $\forall x \exists y R(x, y)$  represents the clause  $\forall x (R(x, A) \vee R(x, B) \vee R(x, C))$ . The decomposition and splitting steps in Algorithm 5 are both easily extended to handle such clauses without loss of lifting (and the base case does not change). However, if universals appear inside existentials, a first-order clause now corresponds to a disjunction of conjunctions of propositional clauses. For example, if the domain is  $\{A, B\}$ ,  $\exists x \forall y (R(x, y) \vee S(x, y))$  represents  $(R(A, A) \vee S(A, A)) \wedge (R(A, B) \vee S(A, B)) \vee (R(B, A) \vee S(B, A)) \wedge (R(B, B) \vee S(B, B))$ . Whether these cases can be handled without loss of lifting remains an open question.

Several optimizations of the basic LWMC procedure in Algorithm 5 can be readily ported from the algorithms PTP generalizes. These optimizations can tremendously improve the performance of LWMC.

**Unit Propagation** When LWMC splits on atom  $A$ , the clauses in the current CNF are resolved with the unit clauses  $A$  and  $\neg A$ . This results in deleting false atoms, which may produce new unit clauses. The idea in unit propagation is to in turn resolve all clauses in the new CNF with all the new unit clauses, and continue to do this until no further unit resolutions are possible. This often produces a much smaller CNF, and is a key component of DPLL that can also be used in LWMC. Other techniques used

in propositional inference that can be ported to LWMC include pure literals, clause learning, clause indexing, and random restarts.<sup>2, 3, 23</sup>

**Caching/Memoization** In a typical inference, LWMC will be called many times on the same subproblems. The solutions of these can be cached when they are computed, and reused when they are encountered again. (Notice that a cache hit only requires identity up to renaming of variables and constants.) This can greatly reduce the time complexity of LWMC, but at the cost of increased space complexity. If the results of all calls to LWMC are cached (full caching), in the worst case LWMC will use exponential space. In practice, we can limit the cache size to the available memory and heuristically prune elements from it when it is full. Thus, by changing the cache size, LWMC can explore various time/space tradeoffs. Caching in LWMC corresponds to both caching in model counting<sup>23</sup> and recursive conditioning<sup>4</sup> and to memoization of common subproofs in theorem proving.

**Knowledge-Based Model Construction** KBMC first uses logical inference to select the subset of the PKB that is relevant to the query, and then propositionalizes the result and performs standard probabilistic inference on it.<sup>25</sup> A similar effect can be obtained in PTP by noticing that in Equation 2 factors that are common to  $Z(K \cup \{(Q, 0)\})$  and  $Z(K)$  cancel out and do not need to be computed. Thus we can modify Algorithm 3 as follows: (i) simplify the PKB by unit propagation starting from evidence atoms, etc.; (ii) drop from the PKB all formulas that have no path of unifiable literals to the query; (iii) pass to LWMC only the remaining formulas, with an initial  $S$  containing the substitutions required for the unifications along the connecting path(s).

We now state two theorems (proofs are provided in the extended version of the paper) which compare the efficiency of PTP and first-order variable elimination (FOVE).<sup>8, 18</sup>

**THEOREM 4.** *PTP can be exponentially more efficient than FOVE.*

**THEOREM 5.** *LWMC with full caching has the same worst-case time and space complexity as FOVE.*

De Salvo Braz's FOVE<sup>8</sup> and lifted BP<sup>24</sup> completely shatter the PKB in advance. This may be wasteful because many of those splits may not be necessary. In contrast, LWMC splits only as needed.

PTP yields new algorithms for several of the inference problems in Figure 1. For example, ignoring weights and replacing products by conjunctions and sums by disjunctions in Algorithm 5 yields a lifted version of DPLL for first-order theorem proving.

Of the standard methods for inference in graphical models, propositional PTP is most similar to recursive conditioning<sup>4</sup> and AND/OR search<sup>9</sup> with context-sensitive decomposition and caching, but applies to arbitrary PKBs, not just Bayesian networks. Also, PTP effectively performs formula-based inference<sup>13</sup> when it splits on one of the auxiliary atoms introduced by Algorithm 2.

PTP realizes some of the benefits of lazy inference for relational models<sup>10</sup> by keeping in lifted form what lazy inference would leave as default.

## 6. APPROXIMATE INFERENCE

LWMC lends itself readily to Monte Carlo approximation, by replacing the sum in the splitting step with a random choice of one of its terms, calling the algorithm many times, and averaging the results. This yields the first lifted sampling algorithm.

We first apply this importance sampling approach<sup>21</sup> to WMC, yielding the MC-WMC algorithm. The two algorithms differ only in the last line. Let  $Q(A|C, W)$  denote the *importance* or *proposal* distribution over  $A$  given the current CNF  $C$  and literal weights  $W$ . Then we return  $\frac{W_A}{Q(A|C, W)}$  MC-WMC( $C|A; W$ ) with probability  $Q(A|C, W)$ , or  $\frac{W_{\neg A}}{Q(\neg A|C, W)}$  MC-WMC( $C|\neg A; W$ ) otherwise. By importance sampling theory<sup>21</sup> and by the law of total expectation, it is easy to show that:

**THEOREM 6.** *If  $Q(A|C, W)$  satisfies  $WMC(C|A; W) > 0 \Rightarrow Q(A|C, W) > 0$  for all atoms  $A$  and its true and false assignments, then the expected value of the quantity output by MC-WMC( $C, W$ ) equals  $WMC(C, W)$ . In other words, MC-WMC( $C, W$ ) yields an unbiased estimate of  $WMC(C, W)$ .*

An estimate of  $WMC(C, W)$  is obtained by running MC-WMC( $C, W$ ) multiple times and averaging the results. By linearity of expectation, the running average is also unbiased. It is well known that the accuracy of the estimate is inversely proportional to its variance.<sup>21</sup> The variance can be reduced by either running MC-WMC more times or by choosing  $Q$  that is as close as possible to the posterior distribution  $P$  (or both). Thus, for MC-WMC to be effective in practice, at each point, given the current CNF  $C$ , we should select  $Q(A|C, W)$  that is as close as possible to the marginal probability distribution of  $A$  w.r.t.  $C$  and  $W$ .

In presence of hard formulas, MC-WMC suffers from the rejection problem<sup>12</sup>: it may return a zero. We can solve this problem by either backtracking when a sample is rejected or by generating samples from the backtrack-free distribution.<sup>12</sup>

Next, we present a lifted version of MC-WMC, which is obtained by replacing the (last line of the) lifted splitting step in LWMC by the following lifted sampling step:

---

**return**  $\frac{n_i W_A^i W_{\neg A}^i}{Q(\Sigma_{A,S}^{(i)})}$  MC-LWMC( $C|\sigma, S_i, W$ )

---

where  $\Sigma_{A,S}^{(i)}$  is sampled from  $Q$ , and  $n_i, t_i, f_i, \sigma$ , and  $S_i$  are as in Proposition 3.

---

In the lifted sampling step, we construct a distribution  $Q$  over the lifted split and sample an element  $\Sigma_{A,S}^{(i)}$  from it. Then we weigh the sampled element w.r.t.  $Q$  and call the algorithm recursively on the CNF conditioned on  $\sigma \in \Sigma_{A,S}^{(i)}$ .

Notice that in the lifted sampling algorithm,  $A$  is a first-order atom and the distribution  $Q(\Sigma_{A,S}^{(i)})$  is defined in a lifted manner. Moreover, since each  $\Sigma_{A,S}^{(i)}$  represents a subset of truth assignments to the groundings of  $A$ , given a ground assignment  $\sigma \in \Sigma_{A,S}^{(i)}$ , the probability of sampling  $\sigma$  is  $Q_G(\sigma) = Q(\Sigma_{A,S}^{(i)})/n_i$ . Therefore, ignoring the decomposition step, MC-LWMC is equivalent to MC-WMC that uses  $Q_G$  to sample a truth assignment to the groundings of  $A$ . In the decomposition step, given a set of identical and disjoint CNFs, we simply sample just one of the CNFs and raise our estimate to the appropriate count. The correctness of this step follows from the fact that given a set  $\{R_1, \dots, R_k\}$  of  $k$

independent and identical random variables and  $m$  random samples  $(r_{1,1}, \dots, r_{1,m})$  of  $R_1$ , the expected value of the product of the random variables equals  $E[R_1]^k$  and  $(\frac{1}{m} \sum_{i=1}^m r_{1,i})^k$  is an asymptotically unbiased estimate of  $E[R_1]^k$ . Therefore, the following theorem immediately follows from Theorem 6.

**THEOREM 7.** *If  $Q(\Sigma_{A,S}^{(i)})$  satisfies  $WMC(C|\sigma; S_i, W) > 0 \Rightarrow Q(\Sigma_{A,S}^{(i)}) > 0$  for all elements  $\Sigma_{A,S}^{(i)}$  of the lifted split of  $A$  for  $C$  under  $S$ , then MC-LWMC( $C, S, W$ ) yields an asymptotically unbiased estimate of  $WMC(C, W)$ .*

Because of the lifted decomposition and sampling steps, the time and space complexity of MC-LWMC is much smaller than that of MC-WMC. As a result, given a time bound the estimate returned by MC-LWMC will be based on a much larger sample size than the one returned by MC-WMC. Since variance goes down (and the accuracy goes up) as we increase the sample size, MC-LWMC has smaller variance (and is potentially more accurate) than MC-WMC.

## 7. EXPERIMENTS

### 7.1. Exact inference

We implemented PTP in C++ and ran all our experiments on a Linux machine with a 2.33 GHz Intel Xeon processor and 2GB of RAM. We used a constraint solver based on forward checking to implement the substitution constraints. We used the following heuristics for splitting. At any point, we prefer an atom which yields the smallest number of recursive calls to LWMC (i.e., an atom that yields maximum lifting). We break ties by selecting an atom that appears in the largest number of ground clauses; this number can be computed using the constraint solver. If it is the same for two or more atoms, we break ties randomly.

We compare the performance of PTP and FOVE<sup>8</sup> on a link prediction PKB (additional experimental results on randomly generated PKBs are presented in the full version of the paper). Link prediction is the problem of determining whether a link exists between two nodes in a network and is an important problem in many domains such as social network analysis and Web mining. We experimented with a simple link prediction PKB consisting of two clauses:  $\text{GoodProf}(x) \wedge \text{GoodStudent}(y) \wedge \text{Advises}(x, y) \Rightarrow \text{FutureProf}(y)$  and  $\text{Coauthor}(x, y) \Rightarrow \text{Advises}(x, y)$ . The PKB has two types of objects: professors ( $x$ ) and students ( $y$ ). Given data on a subset of papers and “goodness” of professors and students, the task is to be predict who advises whom and who is likely to be a professor in the future.

We evaluated the performance of FOVE and PTP along two dimensions: (i) the number of objects and (ii) the amount of evidence. We varied the number of objects from 10 to 1000 and the number of evidence atoms from 10% to 80%.

Figure 2a shows the impact of increasing the number of evidence atoms on the performance of the two algorithms on a link prediction PKB with 100 objects. FOVE runs out of memory (typically after around 20 minutes of runtime) after the percentage of evidence atoms rises above 40%. PTP solves all the problems and is also much faster than FOVE (notice the log-scale on the y-axis). Figure 2b shows the impact of increasing the number of objects on a link prediction PKB

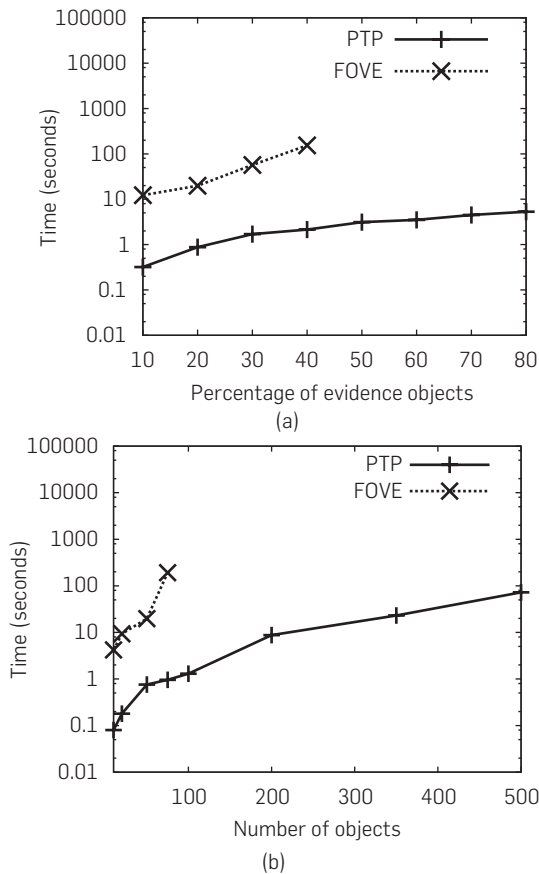
with 20% of the atoms set as observed. We see that FOVE is unable to solve any problems after the number of objects is increased beyond 100 because it runs out of memory. PTP, on the other hand, solves all problems in less than 100s.

## 7.2. Approximate inference

In this subsection, we compare the performance of MC-LWMC, MC-WMC, lifted belief propagation,<sup>24</sup> and MC-SAT<sup>19</sup> on two domains: entity resolution (Cora) and collective classification. The Cora dataset contains 1295 citations to 132 different research papers. The inference task here is to detect duplicate citations, authors, titles, and venues. The collective classification dataset consists of about 3000 query atoms.

Since computing the exact posterior marginals is infeasible in these domains, we used the following evaluation method. We partitioned the data into two equalized sets: evidence set and test set. We then computed the probability of each ground atom in the test set given all atoms in the evidence set using the four inference algorithms. We measure the error using negative log-likelihood of the data according to the inference algorithms (the negative log-likelihood is a sampling approximation of the K-L divergence to the data-generating distribution, shifted by its entropy).

**Figure 2. (a) Impact of increasing the amount of evidence on the time complexity of FOVE and PTP in the link prediction domain. The number of objects in the domain is 100. (b) Impact of increasing the number of objects on the time complexity of FOVE and PTP in the link prediction domain, with 20% of the atoms set as evidence.**



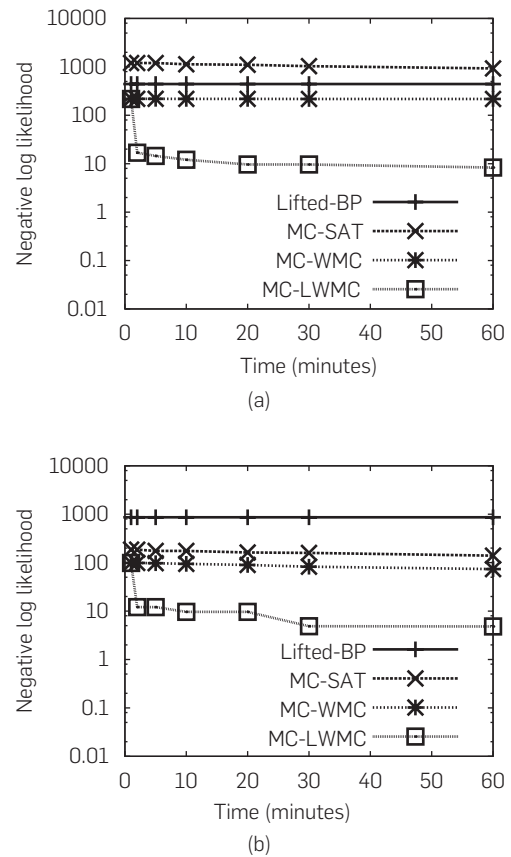
The results, averaged over 10 runs, are shown in Figure 3a and b. The figures show how the log-likelihood of the data varies with time for the four inference algorithms used. We see that MC-LWMC has the lowest negative log-likelihood of all algorithms by a large margin. It significantly dominates MC-WMC in about 2 min of runtime and is substantially superior to both lifted BP and MC-SAT (notice the log scale). This shows the advantages of approximate PTP over lifted BP and ground inference.

## 8. CONCLUSION

Probabilistic theorem proving (PTP) combines theorem proving and probabilistic inference. This paper proposed an algorithm for PTP based on reducing it to lifted weighted model counting, and showed both theoretically and empirically that it has significant advantages compared to previous lifted probabilistic inference algorithms. An implementation of PTP is available in the Alchemy 2.0 system (available at <https://code.google.com/p/alchemy-2/>).

Directions for future research include: extension of PTP to infinite, non-Herbrand first-order logic; new lifted inference rules; theoretical analysis of liftability; porting to PTP more speedup techniques from logical and probabilistic inference; lifted splitting heuristics; better handling of existentials; variational PTP algorithms; better importance distributions; approximate lifting; answering multiple queries simultaneously; applications; etc.

**Figure 3. Negative log-likelihood of the data as a function of time for lifted BP, MC-SAT, MC-WMC, and MC-LWMC on (a) the entity resolution (Cora) and (b) the collective classification domains.**



## Acknowledgments

This research was funded by the ARO MURI grant W911NF-08-1-0242, AFRL contracts FA8750-09-C-0181 and FA8750-14-C-0021, DARPA contracts FA8750-05-2-0283, FA8750-14-C-0005, FA8750-07-D-0185, HR0011-06-C-0025, HR0011-07-C-0060, and NBCH-D030010, NSF grants IIS-0534881 and IIS-0803481, and ONR grant N00014-08-1-0670. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ARO, DARPA, NSF, ONR, or the U.S. Government. ■

## References

1. Bacchus, F. *Representing and Reasoning with Probabilistic Knowledge*. MIT Press, Cambridge, MA, 1990.
2. Bayardo, R.J., Jr., Pehoushek, J.D. Counting models using connected components. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence* (2000), 157–162.
3. Chavira, M., Darwiche, A. On probabilistic inference by weighted model counting. *Artif. Intell.* 172, 6–7 (2008), 772–799.
4. Darwiche, A. Recursive conditioning. *Artif. Intell.* 126, 1–2 (February 2001), 5–41.
5. Davis, M., Logemann, G., Loveland, D. A machine program for theorem proving. *Commun. ACM* 5 (1962) 394–397.
6. Davis, M., Putnam, H. A computing procedure for quantification theory. *J. Assoc. Comput. Mach.* 7, 3 (1960), 201–215.
7. De Raedt, L., Kimmig, A., Toivonen, H. ProbLog: A probabilistic Prolog and its application in link discovery. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence* (2007), 2462–2467.
8. de Salvo Braz, R. Lifted first-order probabilistic inference. PhD thesis, University of Illinois, Urbana-Champaign, IL (2007).
9. Dechter, R., Mateescu, R. AND/OR search spaces for graphical models. *Artif. Intell.* 171, 2–3 (2007), 73–106.
10. Domingos, P., Lowd, D. *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan & Claypool, San Rafael, CA, 2009.
11. Getoor, L., Taskar, B., eds. *Introduction to Statistical Relational Learning*. MIT Press, Cambridge, MA, 2007.
12. Gogate, V., Dechter, R. SampleSearch: Importance sampling in presence of determinism. *Artif. Intell.* 175, 2 (2011), 694–729.
13. Gogate, V., Domingos, P. Formula-based probabilistic inference. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence* (2010), 210–219.
14. Halpern, J. An analysis of first-order logics of probability. *Artif. Intell.* 46 (1990), 311–350.
15. Jha, A., Gogate, V., Meliou, A., Suciu, D. Lifted inference from the other side: The tractable features. In *Proceedings of the Twenty-Fourth Annual Conference on Neural Information Processing Systems* (2010), 973–981.
16. Milch, B., Zettlemoyer, L.S., Kersting, K., Haimes, M., Kaelbling, L.P. Lifted probabilistic inference with counting formulas. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence* (2008), 1062–1068.
17. Nilsson, N. Probabilistic logic. *Artif. Intell.* 28 (1986) 71–87.
18. Poole, D. First-Order probabilistic inference. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence* (2003), 985–991.
19. Poon, H., Domingos, P. Sound and efficient inference with probabilistic and deterministic dependencies. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence* (2006), 458–463.
20. Robinson, J.A. A machine-oriented logic based on the resolution principle. *J. ACM* 12 (1965) 23–41.
21. Rubinstein, R.Y. *Simulation and the Monte Carlo Method*. John Wiley & Sons Inc. Hoboken, NJ, 1981.
22. Sang, T., Beame, P., Kautz, H. Solving Bayesian networks by weighted model counting. In *Proceedings of the Twentieth National Conference on Artificial Intelligence* (2005), 475–482.
23. Sang, T., Beame, P., Kautz, H. Heuristics for fast exact model counting. In *Eighth International Conference on Theory and Applications of Satisfiability Testing* (2005), 226–240.
24. Singla, P., Domingos, P. Lifted first-order belief propagation. In *Lifted first-order belief propagation*. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence* (2008), 1094–1099.
25. Wellman, M., Breese, J.S., Goldman, R.P. From knowledge bases to decision models. *Knowledge Eng. Rev.* 7 (1992), 35–53.

**Vibhav Gogate** (vgogate@hlt.utdallas.edu), The University of Texas at Dallas, Richardson, TX.

**Pedro Domingos** (pedrod@cs.washington.edu), University of Washington Seattle, WA.

© 2016 ACM 0001-0782/16/07 \$15.00

**node**  
SUMMIT  
san francisco

# Everything Node.js

Register today using code **ACM25**  
for 25% off your event pass.  
[www.nodesummit.com/registration](http://www.nodesummit.com/registration)

JULY 27-28, 2016

SAN FRANCISCO

[www.nodesummit.com](http://www.nodesummit.com)

# Technical Perspective

## Mesa Takes Data Warehousing to New Heights

By Sam Madden

LEAVE IT TO Google to make business data processing—among the stodgiest topics in the rather buttoned-up world of database systems—seem cool. The application here involves producing reports over Google's ads infrastructure: Google executives want to see how many ads each Google property is serving, and how profitable they are, and Google's customers want to see how many users are clicking on their ads, how much they are paying, and so on.

At a small scale, solving this problem is straightforward—new ad click and sales data are appended to a database file as they are sent from the processing system, and computing the answer to a particular query over the data involves reading the contents of (“scanning”) the data file to compute a running total of the records in the groups the user is interested in. Making this perform at the scale of Google Ads, where billions of clicks happen per day, is the challenge addressed by the Mesa system described in this following paper.

Fundamentally, the key technique is to employ massive parallelism, both when adding new data and when looking up specific records. The techniques used are largely a collection of best practices developed in the distributed systems and database communities over the last decade, with some clever new ideas thrown in. Some of the highlights from this work include:

► **The use of batch updates and append-only storage.** New data is not added one record at a time, but is aggregated into batches that are sent into Mesa. Instead of merging these batches into the existing storage, these batches are simply written out as additional files that need to be scanned when processing a query. This has the advantage that existing files are never modified, so queries can continue to be executed while new

data is added without worrying about new data being partially read by these existing queries.

► **Massive scale parallel query processing.** Each query is answered by one query processing node, but there can be hundreds or thousands of compute nodes. They can each answer queries independently because of the use of append-only storage: query processors never need to worry that the files they are scanning will change while they are running, and query processors never wait for each other to perform operations.

► **Atomic updates.** Some care is needed to atomically install update batches, such that they are either not seen at all or are seen in their entirety. Mesa labels each update with a unique, monotonically incrementing version number, which is periodically communicated to each query processor. Once a query processor learns of a new version number, it will answer queries up to and including that batch, and is guaranteed that the files containing the batch have been completely written and will not change. This means it can take some time (a few seconds to a minute) for a query processor to see a new update, but this will only result in a slightly stale answer (missing the most recent update), never an answer that is missing some arbitrary subset of updates.

**A natural question is how Mesa compares to existing parallel transactional database systems?**

► **Unusual data model.** Unlike most database systems that simply represent (or “model”) data as a collection of records each with a number of attributes, Mesa allows a programmer to specify a merge function that can be used to combine two records when a record with a duplicate key arrives. This makes it possible, for example, to keep a running total of clicks or revenue for a particular ad or customer. One advantage of this model is that it allows new data to be added without reading the old data first when computing running sums and so on.

A natural question is how Mesa compares to existing parallel transactional database systems? Database systems are optimized for high throughput, but lack several features that are a requirement of the Mesa solution. First, Mesa fits neatly into the elegant modular (layered) software architecture stack Google has built: It runs on top of Colossus (their distributed file system), and provides a substrate on which advanced query processing techniques (like their F1 system) can be built. Layering software this way allows different engineering teams to maintain code, and allows different layers to service multiple clients. Many existing data processing systems are much more monolithic, and would be difficult to integrate into the Google software ecosystem. Second, conventional databases were not built to replicate data across multiple datacenters. Traditional systems (typically) use a single-master approach for fault tolerance, replicating to a (read-only) standby that can take over on a master failure. Such a design will not work well if datacenter failures or network partitions are frequent. 

**Sam Madden** (madden@csail.mit.edu) is a professor of computer science at Massachusetts Institute of Technology, Cambridge, MA.

Copyright held by author.

# Mesa: A Geo-Replicated Online Data Warehouse for Google's Advertising System

By Ashish Gupta, Fan Yang, Jason Govig, Adam Kirsch, Kelvin Chan, Kevin Lai, Shuo Wu, Sandeep Dhoot, Abhilash Rajesh Kumar, Ankur Agiwal, Sanjay Bhansali, Mingsheng Hong, Jamie Cameron, Masood Siddiqi, David Jones, Jeff Shute, Andrey Gubarev, Shivakumar Venkataraman, and Divyakant Agrawal

## Abstract

**Mesa is a highly scalable analytic data warehousing system that stores critical measurement data related to Google's Internet advertising business. Mesa is designed to satisfy a complex and challenging set of user and systems requirements, including near real-time data ingestion and retrieval, as well as high availability, reliability, fault tolerance, and scalability for large data and query volumes. Specifically, Mesa handles petabytes of data, processes millions of row updates per second, and serves billions of queries that fetch trillions of rows per day. Mesa is geo-replicated across multiple datacenters and provides consistent and repeatable query answers at low latency, even when an entire datacenter fails. This paper presents the Mesa system and reports the performance and scale that it achieves.**

## 1. INTRODUCTION

Google runs an extensive advertising platform across multiple channels that serves billions of advertisements (or *ads*) every day to users all over the globe. Detailed information associated with each served ad, such as the targeting criteria, number of impressions and clicks, etc., are recorded and processed in real time. This data is used extensively at Google for different use cases, including reporting, internal auditing, analysis, billing, and forecasting. Advertisers gain fine-grained insights into their advertising campaign performance by interacting with a sophisticated front-end service that issues online and on-demand queries to the underlying data store. Google's internal ad serving platforms use this data in real time, determining budgeting and ad performance to enhance ad serving relevancy. As the Google ad platform continues to expand and as internal and external customers request greater visibility into their advertising campaigns, the demand for more detailed and fine-grained information leads to tremendous growth in the data size. The scale and business critical nature of this data result in unique technical and operational challenges for processing, storing, and querying. The requirements for such a data store are:

*Atomic Updates.* A single user action may lead to multiple updates at the relational data level, affecting thousands of consistent views, defined over a set of metrics (e.g., clicks and cost) across a set of dimensions (e.g., advertiser and country). It must not be possible to query the system in a state where only some of the updates have been applied.

*Consistency and Correctness.* For business and legal reasons, this system must return consistent and correct data. We require strong consistency and repeatable query results even if a query involves multiple datacenters.

*Availability.* The system must not have any single point of failure. There can be no downtime in the event of planned or unplanned maintenance or failures, including outages that affect an entire datacenter or a geographical region.

*Near Real-Time Update Throughput.* The system must support continuous updates, both new rows and incremental updates to existing rows, with the update volume on the order of millions of rows updated per second. These updates should be available for querying consistently across different views and datacenters within minutes.

*Query Performance.* The system must support latency-sensitive users serving live customer reports with very low latency requirements and batch extraction users requiring very high throughput. Overall, the system must support point queries with 99th percentile latency in the hundreds of milliseconds and overall query throughput of trillions of rows fetched per day.

*Scalability.* The system must be able to scale with the growth in data size and query volume. For example, it must support trillions of rows and petabytes of data. The update and query performance must hold even as these parameters grow significantly.

*Online Data and Metadata Transformation.* In order to support new feature launches or change the granularity of existing data, clients often require transformations of the data schema or modifications to existing data values. These changes must not interfere with the normal query and update operations.

Mesa is Google's solution to these technical and operational challenges for business critical data. Mesa is a distributed, replicated, and highly available data processing, storage, and query system for structured data. Mesa ingests data generated by upstream services, aggregates and persists the data internally, and serves the data via user queries. Even though this paper mostly discusses Mesa in the context

The original version of this paper, entitled "Mesa: Geo-Replicated, Near Real-Time, Scalable Warehousing," was published in the *Proceedings of the VLDB Endowment* 7, 12 (Aug. 2014), 1259–1270.

of ads metrics, Mesa is a generic data warehousing solution that satisfies all of the above requirements.

Mesa leverages common Google infrastructure and services, such as Colossus (Google's distributed file system),<sup>7</sup> BigTable,<sup>3</sup> and MapReduce.<sup>6</sup> To achieve storage scalability and availability, data is horizontally partitioned and replicated. Updates may be applied at the granularity of a single table or across many tables. To achieve consistent and repeatable queries during updates, the underlying data is multi-versioned. To achieve update scalability, data updates are batched, assigned a new version number, and periodically (e.g., every few minutes) incorporated into Mesa. To achieve update consistency across multiple data centers, Mesa uses a distributed synchronization protocol based on Paxos.<sup>11</sup>

In contrast, commercial DBMS vendors<sup>4, 14, 17</sup> often address the scalability challenge through specialized hardware and sophisticated parallelization techniques. Internet services companies<sup>1, 12, 16</sup> address this challenge using a combination of new technologies: key-value stores,<sup>3, 8, 13</sup> columnar storage, and the MapReduce programming paradigm. However, many of these systems are designed to support bulk load interfaces to import data and can require hours to run. From that perspective, Mesa is very similar to an OLAP system. Mesa's update cycle is minutes and it continuously processes hundreds of millions of rows. Mesa uses multi-versioning to support transactional updates and queries across tables. A system that is close to Mesa in terms of supporting both dynamic updates and real-time querying of transactional data is Vertica.<sup>10</sup> However, to the best of our knowledge, none of these commercial products or production systems has been designed to manage replicated data across multiple datacenters. Also, none of Google's other in-house data solutions<sup>2, 3, 5, 15</sup> support the data size and update volume required to serve as a data warehousing platform supporting Google's advertising business.

Mesa achieves the required update scale by processing updates in batches. Mesa is, therefore, unique in that application data is redundantly (and independently) processed at all datacenters, while the metadata is maintained using synchronous replication. This approach minimizes the synchronization overhead across multiple datacenters in addition to providing additional robustness in face of data corruption.

## 2. MESA STORAGE SUBSYSTEM

Data in Mesa is continuously generated and is one of the largest and most valuable data sets at Google. Analysis queries on this data can range from simple queries such as, "How many ad clicks were there for a particular advertiser on a specific day?" to a more involved query scenario such as, "How many ad clicks were there for a particular advertiser matching the keyword 'decaf' during the first week of October between 8:00 am and 11:00 am that were displayed on google.com for users in a specific geographic location using a mobile device?"

Data in Mesa is inherently multi-dimensional, capturing all the microscopic facts about the overall performance of Google's advertising platform in terms of different dimensions. These facts typically consist of two types of attributes: *dimensional* attributes (which we call *keys*) and *measure*

attributes (which we call *values*). Since many dimension attributes are hierarchical (and may even have multiple hierarchies, e.g., the *date* dimension can organize data at the day/month/year level or fiscal week/quarter/year level), a single fact may be aggregated in multiple materialized views based on these dimensional hierarchies to enable data analysis using drill-downs and roll-ups. A careful warehouse design requires that the existence of a single fact is consistent across all possible ways the fact is materialized and aggregated.

### 2.1. The data model

In Mesa, data is maintained using *tables*. Each table has a *table schema* that specifies its structure. Specifically, a table schema specifies the *key space*  $K$  for the table and the corresponding *value space*  $V$ , where both  $K$  and  $V$  are sets. The table schema also specifies the *aggregation function*  $F : V \times V \rightarrow V$  which is used to aggregate the values corresponding to the same key. The aggregation function must be associative (i.e.,  $F(F(v_0, v_1), v_2) = F(v_0, F(v_1, v_2))$ ) for any values  $v_0, v_1, v_2 \in V$ . In practice, it is usually also commutative (i.e.,  $F(v_0, v_1) = F(v_1, v_0)$ ), although Mesa does have tables with non-commutative aggregation functions (e.g.,  $F(v_0, v_1) = v_1$  to replace a value). The schema also specifies one or more *indexes* for a table, which are total orderings of  $K$ .

The key space  $K$  and value space  $V$  are represented as tuples of *columns*, each of which has a fixed type (e.g., int32, int64, string, etc.). The schema specifies an associative aggregation function for each individual value column, and  $F$  is implicitly defined as the coordinate-wise aggregation of the value columns, that is:

$$F((x_1, \dots, x_k), (y_1, \dots, y_k)) = (f_1(x_1, y_1), \dots, f_k(x_k, y_k)),$$

where  $(x_1, \dots, x_k), (y_1, \dots, y_k) \in V$  are any two tuples of column values, and  $f_1, \dots, f_k$  are explicitly defined by the schema for each value column.

As an example, Figure 1 illustrates three Mesa tables. All three tables contain ad click and cost metrics (value columns)

Figure 1. Three related Mesa tables.

| Date       | PublisherId | Country | Clicks | Cost |
|------------|-------------|---------|--------|------|
| 2013/12/31 | 100         | US      | 10     | 32   |
| 2014/01/01 | 100         | US      | 205    | 103  |
| 2014/01/01 | 200         | UK      | 100    | 50   |

(a) Mesa table A

| Date       | AdvertiserId | Country | Clicks | Cost |
|------------|--------------|---------|--------|------|
| 2013/12/31 | 1            | US      | 10     | 32   |
| 2014/01/01 | 1            | US      | 5      | 3    |
| 2014/01/01 | 2            | UK      | 100    | 50   |
| 2014/01/01 | 2            | US      | 200    | 100  |

(b) Mesa table B

| AdvertiserId | Country | Clicks | Cost |
|--------------|---------|--------|------|
| 1            | US      | 15     | 35   |
| 2            | UK      | 100    | 50   |
| 2            | US      | 200    | 100  |

(c) Mesa table C

broken down by various attributes, such as the date of the click, the advertiser, the publisher website that showed the ad, and the country (key columns). The aggregation function used for both value columns is *SUM*. All metrics are consistently represented across the three tables, assuming the same underlying events have updated data in all these tables. Figure 1 is a simplified view of Mesa's table schemas. In production, Mesa contains over a thousand tables, many of which have hundreds of columns, using various aggregation functions.

## 2.2. Updates and queries

To achieve high update throughput, Mesa applies updates in batches. The update batches themselves are produced by an upstream system outside of Mesa, typically at a frequency of every few minutes (smaller and more frequent batches would imply lower update latency, but higher resource consumption). Formally, an *update* to Mesa specifies a *version number*  $n$  (sequentially assigned from 0) and a set of *rows* of the form (table name, key, value). Each update contains at most one aggregated value for every (table name, key).

A *query* to Mesa consists of a version number  $n$  and a predicate  $P$  on the key space. The response contains one row for each key matching  $P$  that appears in some update with version between 0 and  $n$ . The value for a key in the response is the aggregate of all values for that key in those updates. Mesa actually supports more complex query functionality than this, but all of that can be viewed as pre-processing and post-processing with respect to this primitive.

As an example, Figure 2 shows two updates corresponding to tables defined in Figure 1 that, when aggregated, yield tables A, B, and C. To maintain table consistency (as discussed in Section 2.1), each update contains consistent rows for the two tables, A and B. Mesa computes

the updates to table C automatically, because they can be derived directly from the updates to table B. Conceptually, a single update including both the *AdvertiserId* and *PublisherId* attributes could also be used to update all three tables, but that could be expensive, especially in more general cases where tables have many attributes (e.g., due to a cross product).

Note that table C corresponds to a materialized view of the following query over table B: `SELECT SUM(Clicks), SUM(Cost) GROUP BY AdvertiserId, Country`. This query can be represented directly as a Mesa table because the use of *SUM* in the query matches the use of *SUM* as the aggregation function for the value columns in table B. Mesa restricts materialized views to use the same aggregation functions for metric columns as the parent table.

To enforce update atomicity, Mesa uses a multi-versioned approach. Mesa applies updates in order by version number, ensuring atomicity by always incorporating an update entirely before moving on to the next update. Users can never see any effects from a partially incorporated update.

The strict ordering of updates has additional applications beyond atomicity. Indeed, the aggregation functions in the Mesa schema may be non-commutative, such as in the standard key-value store use case where a (key, value) update completely overwrites any previous value for the key. More subtly, the ordering constraint allows Mesa to support use cases where an incorrect fact is represented by an inverse action. In particular, Google uses online fraud detection to determine whether ad clicks are legitimate. Fraudulent clicks are offset by *negative facts*. For example, there could be an update version 2 following the updates in Figure 2 that contains negative clicks and costs, corresponding to marking previously processed ad clicks as illegitimate. By enforcing strict ordering of updates, Mesa ensures that a negative fact can never be incorporated before its positive counterpart.

## 2.3. Versioned data management

Versioned data plays a crucial role in both update and query processing in Mesa. However, it presents multiple challenges. First, given the aggregatable nature of ads statistics, storing each version independently is very expensive from the storage perspective. The aggregated data can typically be much smaller. Second, going over all the versions and aggregating them at query time is also very expensive and increases the query latency. Third, naïve pre-aggregation of all versions on every update can be prohibitively expensive.

To handle these challenges, Mesa pre-aggregates certain versioned data and stores it using *deltas*, each of which consists of a set of rows (with no repeated keys) and a *delta version* (or, more simply, a *version*), represented by  $[V_1, V_2]$ , where  $V_1$  and  $V_2$  are update version numbers and  $V_1 \leq V_2$ . We refer to deltas by their versions when the meaning is clear. The rows in a delta  $[V_1, V_2]$  correspond to the set of keys that appeared in updates with version numbers between  $V_1$  and  $V_2$  (inclusively). The value for each such key is the aggregation of its values in those updates. Updates are incorporated into Mesa as *singleton deltas* (or, more simply, *singletons*).

**Figure 2. Two Mesa updates.**

| Date       | PublisherId | Country | Clicks | Cost |
|------------|-------------|---------|--------|------|
| 2013/12/31 | 100         | US      | +10    | +32  |
| 2014/01/01 | 100         | US      | +150   | +80  |
| 2014/01/01 | 200         | UK      | +40    | +20  |

(a) Update version 0 for Mesa table A

| Date       | AdvertiserId | Country | Clicks | Cost |
|------------|--------------|---------|--------|------|
| 2013/12/31 | 1            | US      | +10    | +32  |
| 2014/01/01 | 2            | UK      | +40    | +20  |
| 2014/01/01 | 2            | US      | +150   | +80  |

(b) Update version 0 for Mesa table B

| Date       | PublisherId | Country | Clicks | Cost |
|------------|-------------|---------|--------|------|
| 2014/01/01 | 100         | US      | +55    | +23  |
| 2014/01/01 | 200         | UK      | +60    | +30  |

(c) Update version 1 for Mesa table A

| Date       | AdvertiserId | Country | Clicks | Cost |
|------------|--------------|---------|--------|------|
| 2013/01/01 | 1            | US      | +5     | +3   |
| 2014/01/01 | 2            | UK      | +60    | +30  |
| 2014/01/01 | 2            | US      | +50    | +20  |

(d) Update version 1 for Mesa table B

The delta version  $[V_1, V_2]$  for a singleton corresponding to an update with version number  $n$  is denoted by setting  $V_1 = V_2 = n$ .

A delta  $[V_1, V_2]$  and another delta  $[V_2 + 1, V_3]$  can be aggregated to produce the delta  $[V_1, V_3]$ , simply by merging row keys and aggregating values accordingly. (As discussed in Section 2.4, the rows in a delta are sorted by key, and, therefore, two deltas can be merged in linear time.) The correctness of this computation follows from associativity of the aggregation function  $F$ . Notably, correctness does not depend on commutativity of  $F$ , as whenever Mesa aggregates two values for a given key, the delta versions are always of the form  $[V_1, V_2]$  and  $[V_2 + 1, V_3]$ , and the aggregation is performed in the increasing order of versions.

Mesa allows users to query at a particular version for only a limited time period (e.g., 24 hours). This implies that versions that are older than this time period can be aggregated into a *base delta* (or, more simply, a *base*) with version  $[0, B]$  for some *base version*  $B \geq 0$ , and after that any other deltas  $[V_1, V_2]$  with  $0 \leq V_1 \leq V_2 \leq B$  can be deleted. This process is called *base compaction*, and Mesa performs it concurrently and asynchronously with respect to other operations (e.g., incorporating updates and answering queries).

Note that for compaction purposes, the time associated with an update version is the time that version was generated, which is independent of any time series information that may be present in the data. For example, for the Mesa tables in Figure 1, the data associated with 2014/01/01 is never removed. However, Mesa may reject a query to the particular depicted version after some time. The date in the data is just another attribute and is opaque to Mesa.

With base compaction, to answer a query for version number  $n$ , we could aggregate the base delta  $[0, B]$  with all singleton deltas  $[B + 1, B + 1]$ ,  $[B + 2, B + 2]$ ,  $\dots$ ,  $[n, n]$ , and then return the requested rows. Even though we run base compaction frequently (e.g., every day), the number of singletons can still easily approach hundreds (or even a thousand), especially for update intensive tables. In order to support more efficient query processing, Mesa maintains a set of *cumulative deltas*  $D$  of the form  $[U, V]$  with  $B < U < V$  through a process called *cumulative compaction*. These deltas can be used to find a *spanning set* of deltas  $\{[0, B], [B + 1, V_1], [V_1 + 1, V_2], \dots, [V_k + 1, n]\}$  for a version  $n$  that requires significantly less aggregation than simply using the singletons. Of course, there is a storage and processing cost associated with the cumulative deltas, but that cost is amortized over all operations (particularly queries) that are able to use those deltas instead of singletons.

The *delta compaction policy* determines the set of deltas maintained by Mesa at any point in time. Its primary purpose is to balance the processing that must be done for a query, the latency with which an update can be incorporated into a Mesa delta, and the processing and storage costs associated with generating and maintaining deltas. More specifically, the delta policy determines: (i) what deltas (excluding the singleton) must be generated prior to allowing an update

version to be queried (synchronously inside the update path, slowing down updates at the expense of faster queries), (ii) what deltas should be generated asynchronously outside of the update path, and (iii) when a delta can be deleted.

An example of delta compaction policy is the *two-level* policy illustrated in Figure 3. Under this example policy, at any point in time there is a base delta  $[0, B]$ , cumulative deltas with versions  $[B + 1, B + 10]$ ,  $[B + 1, B + 20]$ ,  $[B + 1, B + 30]$ ,  $\dots$ , and singleton deltas for every version greater than  $B$ . Generation of the cumulative  $[B + 1, B + 10x]$  begins asynchronously as soon as a singleton with version  $B + 10x$  is incorporated. A new base delta  $[0, B']$  is computed approximately every day, but the new base cannot be used until the corresponding cumulative deltas relative to  $B'$  are generated as well. When the base version  $B$  changes to  $B'$ , the policy deletes the old base, old cumulative deltas, and any singletons with versions less than or equal to  $B'$ . A query then involves the base, one cumulative, and a few singletons, reducing the amount of work done at query time. Mesa currently uses a variation of the two-level delta policy in production that uses multiple levels of cumulative deltas.<sup>9</sup>

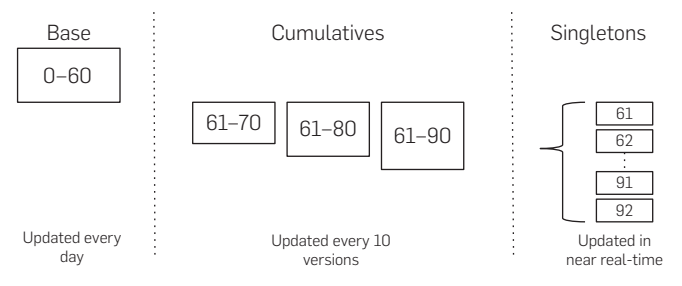
## 2.4. Physical data and index formats

Mesa deltas are created and deleted based on the delta compaction policy. Once a delta is created, it is immutable, and therefore there is no need for its physical format to efficiently support incremental modification.

The immutability of Mesa deltas allows them to use a fairly simple physical format. The primary requirements are only that the format must be space efficient, as storage is a major cost for Mesa, and that it must support fast seeking to a specific key, because a query often involves seeking into several deltas and aggregating the results across keys. To enable efficient seeking using keys, each Mesa table has one or more *table indexes*. Each table index has its own copy of the data that is sorted according to the index's order.

The details of the format itself are somewhat technical, so we focus only on the most important aspects. The rows in a delta are stored in sorted order in *datafiles* of bounded size (to optimize for filesystem file size constraints). The rows themselves are organized into *row blocks*, each of which is individually transposed and compressed. The transposition lays out the data by column instead of by row to allow for better compression. Since storage is a

**Figure 3. A two-level delta compaction policy.**



major cost for Mesa and decompression performance on read/query significantly outweighs the compression performance on write, we emphasize the compression ratio and read/decompression times over the cost of write/compression times when choosing the compression algorithm.

Mesa also stores an *index file* corresponding to each data file. (Recall that each data file is specific to a higher-level table index.) An index entry contains the *short key* for the row block, which is a fixed size prefix of the first key in the row block, and the offset of the compressed row block in the data file. A naïve algorithm for querying a specific key is to perform a binary search on the index file to find the range of row blocks that may contain a short key matching the query key, followed by a binary search on the compressed row blocks in the data files to find the desired key.

### 3. MESA SYSTEM ARCHITECTURE

Mesa is built using common Google infrastructure and services, including BigTable<sup>3</sup> and Colossus.<sup>7</sup> Mesa runs in multiple datacenters, each of which runs a single *instance*. We start by describing the design of an instance. Then we discuss how those instances are integrated to form a full multi-datacenter Mesa deployment.

#### 3.1. Single datacenter instance

Each Mesa instance is composed of two subsystems: update/maintenance and querying. These subsystems are decoupled, allowing them to scale independently. All persistent metadata is stored in BigTable and all data files are stored in Colossus. No direct communication is required between the two subsystems for operational correctness.

**Update/maintenance subsystem.** The update and maintenance subsystem performs all necessary operations to ensure the data in the local Mesa instance is correct, up to date, and optimized for querying. It runs various background operations such as loading updates, performing table compaction, applying schema changes, and running table checksums. These operations are managed and performed by a collection of components known as the *controller/worker framework*, illustrated in Figure 4.

The *controller* determines the work that needs to be done and manages all *table metadata*, which it persists in the *metadata BigTable*. The table metadata consists of detailed state and operational metadata for each table,

including entries for all delta files and update versions associated with the table, the delta compaction policy assigned to the table, and accounting entries for current and previously applied operations broken down by the operation type.

The controller can be viewed as a large-scale table meta-data cache, work scheduler, and work queue manager. The controller does not perform any actual table data manipulation work; it only schedules work and updates the meta-data. At startup, the controller loads table metadata from a BigTable, which includes entries for all tables assigned to the local Mesa instance. For every known table, it subscribes to a metadata feed to listen for table updates. This subscription is dynamically updated as tables are added and dropped from the instance. New update metadata received on this feed is validated and recorded. The controller is the exclusive writer of the table metadata in the BigTable.

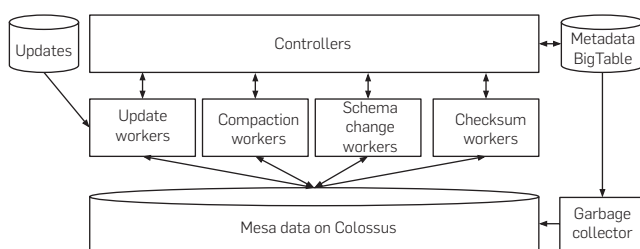
The controller maintains separate internal queues for different types of data manipulation work (e.g., incorporating updates, delta compaction, schema changes, and table checksums). For operations specific to a single Mesa instance, such as incorporating updates and delta compaction, the controller determines what work to queue. Work that requires globally coordinated application or global synchronization, such as schema changes and table checksums, are initiated by other components that run outside the context of a single Mesa instance. For these tasks, the controller accepts work requests by RPC and inserts these tasks into the corresponding internal work queues.

Worker components are responsible for performing the data manipulation work within each Mesa instance. Mesa has a separate set of worker pools for each task type, allowing each worker pool to scale independently. Mesa uses an in-house worker pool scheduler that scales the number of workers based on the percentage of idle workers available. A worker can process a large task using MapReduce.<sup>9</sup>

Each idle worker periodically polls the controller to request work for the type of task associated with its worker type until valid work is found. Upon receiving valid work, the worker validates the request, processes the retrieved work, and notifies the controller when the task is completed. Each task has an associated maximum ownership time and a periodic lease renewal interval to ensure that a slow or dead worker does not hold on to the task forever. The controller is free to reassign the task if either of the above conditions could not be met; this is safe because the controller will only accept the task result from the worker to which it is assigned. This ensures that Mesa is resilient to worker failures. A *garbage collector* runs continuously to delete files left behind due to worker crashes.

Since the controller/worker framework is only used for update and maintenance work, these components can restart without impacting external users. Also, the controller itself is sharded by table, allowing the framework to scale. In addition, the controller is stateless – all state information is maintained consistently in the BigTable. This ensures that

Figure 4. Mesa's controller/worker framework.



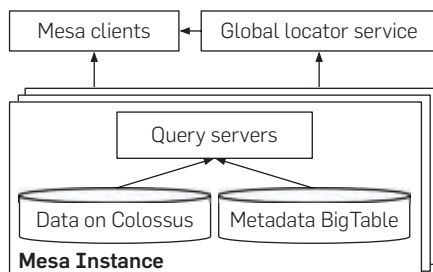
Mesa is resilient to controller failures, since a new controller can reconstruct the state prior to the failure from the meta-data in the BigTable.

**Query subsystem.** Mesa's query subsystem consists of *query servers*, illustrated in Figure 5. These servers receive user queries, look up table metadata, determine the set of files storing the required data, perform on-the-fly aggregation of this data, and convert the data from the Mesa internal format to the client protocol format before sending the data back to the client. Mesa's query servers provide a limited query engine with basic support for server-side conditional filtering and "group by" aggregation. Higher-level database engines such as MySQL and F1 use these primitives to provide richer SQL functionality such as join queries.

Mesa clients have vastly different requirements and performance characteristics. In some use cases, Mesa receives queries directly from interactive reporting front-ends, which have very strict low-latency requirements. These queries are usually small but must be fulfilled almost immediately. Mesa also receives queries from large extraction-type workloads, such as offline daily reports, that send millions of requests and fetch billions of rows per day. These queries require high throughput and are typically not latency sensitive (a few seconds/minutes of latency is acceptable). Mesa ensures that these latency and throughput requirements are met by requiring workloads to be labeled appropriately and then using those labels in isolation and prioritization mechanisms in the query servers.

The query servers for a single Mesa instance are organized into multiple *sets*, each of which is collectively capable of serving all tables known to the controller. By using multiple sets of query servers, it is easier to perform query server updates (e.g., binary releases) without unduly impacting clients, who can automatically failover to another set in the same (or even a different) Mesa instance. Within a set, each query server is in principle capable of handling a query for any table. However, for performance reasons, Mesa prefers to direct queries over similar data (e.g., all queries over the same table) to a subset of the query servers. This technique allows Mesa to provide strong latency guarantees by allowing for effective query server in-memory pre-fetching and caching of data stored in Colossus, while also allowing for excellent overall throughput by balancing load

Figure 5. Mesa's query processing framework.



across the query servers. On startup, each query server registers the list of tables it actively caches with a *global locator service*, which is then used by clients to discover query servers.

There are many optimizations involved in query processing.<sup>9</sup> One important class is skipping unnecessary rows or deltas. Another is allowing a failed query to resume mid-stream from another query server, possibly in another data center.

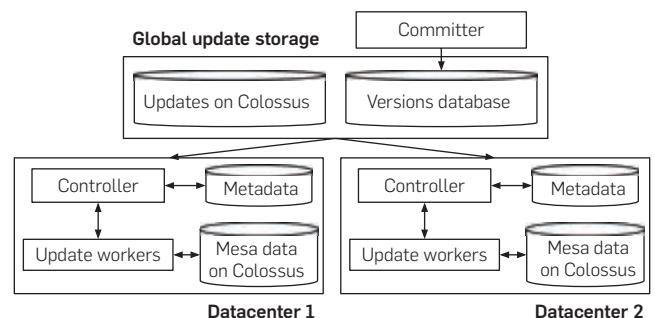
### 3.2. Multi-datacenter deployment

Mesa is deployed in multiple geographical regions in order to provide high availability. Each instance is independent and stores a separate copy of the data. In this section, we discuss the global aspects of Mesa's architecture.

**Consistent update mechanism.** All tables in Mesa are multi-versioned, allowing Mesa to continue to serve consistent data from previous states while new updates are being processed. An upstream system generates the update data in batches for incorporation by Mesa, typically once every few minutes. As illustrated in Figure 6, Mesa's *committer* is responsible for coordinating updates across all Mesa instances worldwide, one version at a time. The committer assigns each update batch a new version number and publishes all metadata associated with the update (e.g., the locations of the files containing the update data) to the *versions database*, a globally replicated and consistent data store build on top of the Paxos<sup>11</sup> consensus algorithm. The committer itself is stateless, with instances running in multiple datacenters to ensure high availability.

Mesa's controllers listen to the changes to the versions database to detect the availability of new updates, assign the corresponding work to update workers, and report successful incorporation of the update back to the versions database. The committer continuously evaluates if *commit criteria* are met (specifically, whether the update has been incorporated by a sufficient number of Mesa instances across multiple geographical regions). The committer enforces the commit criteria across all tables in the update. This property is essential for maintaining consistency of related tables (e.g., a Mesa table that is a materialized view over another Mesa table). When the commit criteria are met, the committer declares the update's version number

Figure 6. Update processing in a multi-datacenter Mesa deployment.



to be the new *committed version*, storing that value in the versions database. New queries are always issued against the committed version.

Mesa's update mechanism design has interesting performance implications. First, since all new queries are issued against the committed version and updates are applied in batches, Mesa does not require any locking between queries and updates. Second, all update data is incorporated asynchronously by the various Mesa instances, with only meta-data passing through the synchronously replicated Paxos-based versions database. Together, these two properties allow Mesa to simultaneously achieve very high query and update throughputs.

**New Mesa instances.** As Google builds new datacenters and retires older ones, we need to bring up new Mesa instances. To bootstrap a new Mesa instance, we use a peer-to-peer load mechanism. Mesa has a special *load worker* (similar to other workers in the controller/worker framework) that copies a table from another Mesa instance to the current one. Mesa then uses the update workers to catch up to the latest committed version for the table before making it available to queries. During bootstrapping, we do this to load all tables into a new Mesa instance. Mesa also uses the same peer-to-peer load mechanism to recover from table corruptions.

#### 4. EXPERIENCES AND LESSONS LEARNED

In this section, we briefly highlight the key lessons we have learned from building a large-scale data warehousing system over the past few years. We provide a list that is by no means exhaustive.

*Cloud Computing and Layered Architecture.* Mesa is a distributed cloud system that is built on top of other distributed cloud systems, such as Colossus and BigTable. Mesa's distributed architecture is clearly critical to its scalability. The layered approach is also crucial, as it has allowed us to focus on the key aspects of Mesa, delegating complexity to other systems where appropriate. Often this delegation has a performance cost, but we have found that we can leverage other aspects of the architecture to compensate. For example, when we built Mesa we migrated data from high-performance local disks to Colossus, compensating for the increased seek times by having query servers aggressively pre-fetch data with a lot of parallelism.

*Application Level Assumptions.* One has to be very careful about making strong assumptions about applications while designing large-scale infrastructure. For example, when designing Mesa's predecessor system, we made an assumption that schema changes would be very rare. This assumption turned out to be wrong, and we found that Mesa needed to support online schema changes that do not block either queries or updates, often without making extra copies of the data.<sup>9</sup> Due to the constantly evolving nature of a live enterprise, products, services, and applications are in constant flux. Furthermore, new applications come on board either organically or due to acquisitions of other companies that need to be supported. In summary, the design should be as

general as possible with minimal assumptions about current and future applications.

*Geo-Replication.* Although we support geo-replication in Mesa for high data and system availability, we have also seen added benefit in terms of our day-to-day operations. In Mesa's predecessor system, when there was a planned maintenance outage of a datacenter, we had to perform a laborious operations drill to migrate a  $24 \times 7$  operational system to another datacenter. Today, such planned outages, which are fairly routine, have minimal impact on Mesa.

*Data Corruption and Component Failures.* Data corruption and component failures are a major concern for systems at the scale of Mesa. Data corruptions can arise for a variety of reasons and it is extremely important to have the necessary tools in place to prevent and detect them. Similarly, a faulty component such as a floating-point unit on one machine can be extremely hard to diagnose. Due to the dynamic nature of the allocation of cloud machines to Mesa, it is highly uncertain whether such a machine is consistently active. Furthermore, even if the machine with the faulty unit is actively allocated to Mesa, its usage may cause only intermittent issues. Overcoming such operational challenges remains an open problem, but we discuss some techniques used by Mesa in Ref.<sup>9</sup>

*Testing and Incremental Deployment.* Mesa is a large, complex, critical, and continuously evolving system. Simultaneously maintaining new feature velocity and the health of the production system is a crucial challenge. Fortunately, we have found that by combining some standard engineering practices with Mesa's overall fault-tolerant architecture and resilience to data corruptions, we can consistently deliver major improvements to Mesa with minimal risk. Some of the techniques we use are: unit testing, private developer Mesa instances that can run with a small fraction of production data, and a shared testing environment that runs with a large fraction of production data from upstream systems. We are careful to incrementally deploy new features across Mesa instances. For example, when deploying a high-risk feature, we might deploy it to one instance at a time. Since Mesa has measures to detect data inconsistencies across multiple datacenters (along with thorough monitoring and alerting on all components), we find that we can detect and debug problems quickly.

#### 5. MESA PRODUCTION METRICS

In this section, we report update and query processing performance metrics for Mesa's production deployment. We show the metrics over a 7-day period to demonstrate both their variability and stability. We also show system growth metrics over a multi-year period to illustrate how the system scales to support increasing data sizes with linearly increasing resource requirements, while ensuring the required query performance. Overall, Mesa is highly decentralized and replicated over multiple datacenters, using hundreds to thousands of machines at each datacenter for both update and query processing. Although we do not report the proprietary details of our deployment,

the architectural details that we do provide are comprehensive and convey the highly distributed, large-scale nature of the system.

### 5.1. Update processing

Figure 7 illustrates Mesa update performance for one data source over a 7-day period. Mesa supports hundreds of concurrent update data sources. For this particular data source, on average, Mesa reads 30–60 megabytes of compressed data per second, updating 3–6 million distinct rows and adding about 300 thousand new rows. The data source generates updates in batches about every 5 min, with median and 95th percentile Mesa commit times of 54 and 211 s. Mesa maintains this update latency, avoiding update backlog by dynamically scaling resources.

### 5.2. Query processing

Figure 8 illustrates Mesa’s query performance over a 7-day period for tables from the same data source as above. Mesa executed more than 500 million queries per day for those tables, returning 1.7–3.2 trillion rows. The nature of these production queries varies greatly, from simple point look-ups to large range scans. We report their average and 99th percentile latencies, which show that Mesa answers most queries within tens to hundreds of milliseconds. The large

**Figure 7. Update performance for a single data source over a 7-day period.**

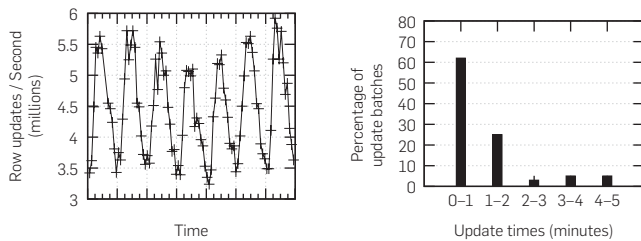
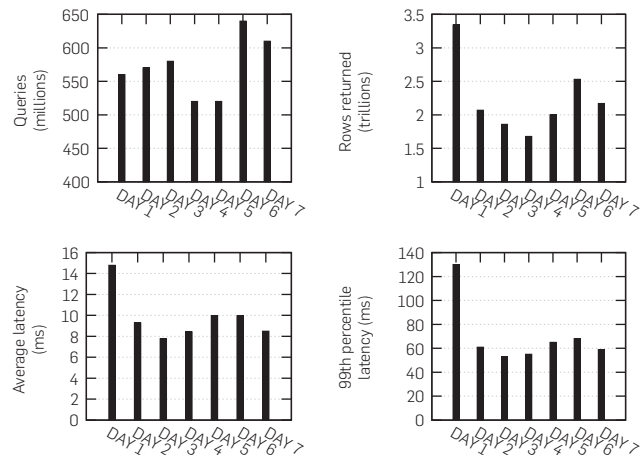


**Figure 8. Query performance metrics for a single data source over a 7-day period.**



difference between the average and tail latencies is driven by multiple factors, including the type of query, the contents of the query server caches, transient failures and retries at various layers of the cloud architecture, and even the occasional slow machine.

In Figure 9, we report the scalability characteristics of Mesa’s query servers. Mesa’s design allows components to independently scale with augmented resources. In this evaluation, we measure the query throughput as the number of servers increases from 4 to 128. This result establishes linear scaling of Mesa’s query processing.

### 5.3. Growth

Figure 10 illustrates the data and CPU usage growth in Mesa over a 24-month period for one of our largest production data sets. Total data size increased almost 500%, driven by update rate (which increased by over 80%) and the addition of new tables, indexes, and materialized views. CPU usage increased similarly, driven primarily by the cost of periodically rewriting data during base compaction, but also affected by one-off computations such as schema changes, as well as optimizations that were deployed over time. Figure 10 also includes fairly stable latency measurements by a monitoring tool that continuously issues synthetic point queries to Mesa that bypass the query server caches. In fact, throughout this period, Mesa answered user point queries with latencies consistent with those shown in Figure 8, while maintaining a similarly high rate of rows returned.

## 6. CONCLUSION

In this paper, we present an end-to-end design and implementation of a geo-replicated, near real-time, scalable data warehousing system called Mesa. The engineering design of Mesa leverages foundational research ideas in the areas of databases and distributed systems. In particular, Mesa supports online queries and updates while providing

**Figure 9. Scalability of query throughput.**

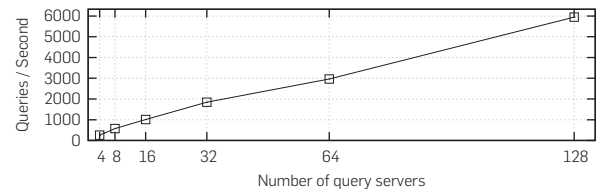
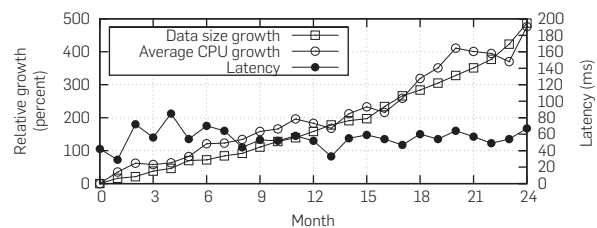


**Figure 10. Growth and latency metrics over a 24-month period.**



strong consistency and transactional correctness guarantees. It achieves these properties using a batch-oriented interface, guaranteeing atomicity of updates by introducing transient versioning of data that eliminates the need for lock-based synchronization of query and update transactions. Mesa is geo-replicated across multiple datacenters for increased fault-tolerance. Finally, within each datacenter, Mesa's controller/worker framework allows it to distribute work and dynamically scale the required computation over a large number of machines to provide high scalability.

## Acknowledgments

We would like to thank everyone who has served on the Mesa team, including former team members Karthik Lakshminarayanan, Sanjay Agarwal, Sivasankaran Chandrasekar, Justin Tolmer, Chip Turner, and Michael Ballbach, for their substantial contributions to the design and development of Mesa. We are also grateful to Sridhar Ramaswamy for providing strategic vision and guidance to the Mesa team. Finally, we thank the anonymous reviewers, whose feedback significantly improved the paper.

## References

1. Abouzeid, A., Bajda-Pawlikowski, K., et al. HadoopDB: An architectural hybrid of MapReduce and DBMS technologies for analytical workloads. *PVLDB* 2, 1 (2009), 922–933.
2. Baker, J., Bond, C., et al. Megastore: Providing scalable, highly available storage for interactive services. In *CIDR* (2011), 223–234.
3. Chang, F., Dean, J., et al. Bigtable: A distributed storage system for structured data. In *OSDI* (2006), 205–218.
4. Cohen, J., Eshleman, J., et al. Online expansion of largescale data warehouses. *PVLDB* 4, 12 (2011), 1249–1259.
5. Corbett, J.C., Dean, J., et al. Spanner: Google's globally-distributed database. In *OSDI* (2012), 251–264.
6. Dean, J., Ghemawat, S. MapReduce: Simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.
7. Fikes, A. Storage architecture and challenges. <http://goo.gl/pF6kmz>, 2010.
8. Glendenning, L., Beschastnikh, I., et al. Scalable consistency in scatter. In *SOSP* (2011), 15–28.
9. Gupta, A., Yang, F., et al. Mesa: Geo-replicated, near real-time, scalable data warehousing. In *Vldb* (2014).
10. Lamb, A., Fuller, M., et al. The Vertica analytic database: C-Store 7 years later. *PVLDB* 5, 12 (2012), 1790–1801.
11. Lamport, L. The part-time parliament. *ACM Trans. Comput. Syst.* 16, 2 (1998), 133–169.
12. Lee, G., Lin, J., et al. The unified logging infrastructure for data analytics at Twitter. *PVLDB* 5, 12 (2012), 1771–1780.
13. Project Voldemort: A Distributed Database. <http://www.project-voldemort.com/voldemort/>.
14. SAP HANA. <http://www.saphana.com/welcome>.
15. Shute, J., Vingralek, R., et al. F1: A distributed SQL database that scales. *PVLDB* 6, 11 (2013), 1068–1079.
16. Thusoo, A., Shao, Z., et al. Data warehousing and analytics infrastructure at Facebook. In *SIGMOD* (2010), 1013–1020.
17. Weiss, R. A technical overview of the oracle exadata database machine and exadata storage server. *Oracle White Paper: Oracle Corporation, Redwood Shores*, 2012.

Ashish Gupta, Fan Yang, Jason Govig, Adam Kirsch, Kelvin Chan, Kevin Lai, Shuo Wu, Sandeep Dhoot, Abhilash Rajesh Kumar, Ankur Agiwal, Sanjay Bhansali, Mingsheng Hong,

Jamie Cameron, Masood Siddiqi, David Jones, Jeff Shute, Andrey Gubarev, Shivakumar Venkataraman, and Divyakant Agrawal, Google, Inc., Mountain View, CA.

Copyright held by authors/owner.

# World-Renowned Journals from ACM

ACM publishes over 50 magazines and journals that cover an array of established as well as emerging areas of the computing field. IT professionals worldwide depend on ACM's publications to keep them abreast of the latest technological developments and industry news in a timely, comprehensive manner of the highest quality and integrity. For a complete listing of ACM's leading magazines & journals, including our renowned Transaction Series, please visit the ACM publications homepage: [www.acm.org/pubs](http://www.acm.org/pubs).

## ACM Transactions on Interactive Intelligent Systems



**ACM Transactions on Interactive Intelligent Systems (TIIS).** This quarterly journal publishes papers on research encompassing the design, realization, or evaluation of interactive systems incorporating some form of machine intelligence.

## ACM Transactions on Computation Theory



**ACM Transactions on Computation Theory (ToCT).** This quarterly peer-reviewed journal has an emphasis on computational complexity, foundations of cryptography and other computation-based topics in theoretical computer science.

PLEASE CONTACT ACM MEMBER SERVICES TO PLACE AN ORDER  
 Phone: 1.800.342.6626 (U.S. and Canada)  
 +1.212.626.0500 (Global)  
 Fax: +1.212.944.1318  
 (Hours: 8:30am–4:30pm, Eastern Time)  
 Email: [acmhelp@acm.org](mailto:acmhelp@acm.org)  
 Mail: ACM Member Services  
 General Post Office  
 PO Box 30777  
 New York, NY 10087-0777 USA



Association for  
Computing Machinery

*Advancing Computing as a Science & Profession*

[www.acm.org/pubs](http://www.acm.org/pubs)

# CAREERS

## Purdue University Professor of Practice in Computer Science

The Department of Computer Science at Purdue University is soliciting applications for Professor of Practice positions at the Assistant, Associate, or Full Professor level to begin Fall 2016. These are newly created positions offering three- to five-year appointments that are renewable based on satisfactory performance for faculty with primary responsibilities in teaching and service. Applicants should hold a PhD in computer science or a related field, or a Master's degree in computer science or a related discipline and commensurate experience in teaching or industry. Applicants should be committed to excellence in teaching, and should have the ability to teach a broad collection of core courses in the undergraduate curriculum. Applicants will also be expected to develop and supervise project courses for undergraduates. Review of applications and candidate interviews will begin on May 5, 2016, and will continue until the positions are filled.

The Department of Computer Science offers a stimulating and nurturing educational environment with thriving undergraduate and graduate

programs and active research programs in most areas of computer science. Additional information about the department is available at <http://www.cs.purdue.edu>. Salary and benefits will be competitive.

Applicants should hold a PhD in computer science or a related field, or a Master's degree in computer science or a related discipline and commensurate experience in teaching or industry. Applicants should be committed to excellence in teaching, and should have the ability to teach a broad collection of core courses in the undergraduate curriculum. Applicants will also be expected to develop and supervise project courses for undergraduates.

**Applicants are strongly encouraged to apply online at <https://hiring.science.purdue.edu>.** Alternately hard-copy applications may be sent to: Professor of Practice Search Chair, Department of Computer Science, 305 N. University St., Purdue University, West Lafayette IN 47907. A background check will be required for employment.

Purdue University is an EEO/AA employer. All individuals, including minorities, women, individuals with disabilities, and veterans are encouraged to apply.

The Hasso Plattner Institute (HPI) in Potsdam, Germany, is a university excellence center in Computer Science and IT Systems Engineering.

Annually, the Institute's Research School grants  
**10 Ph.D. and Postdoc Scholarships**

With its interdisciplinary and international structure, the Research School interconnects the HPI research groups as well as its branches at University of Cape Town, Technion, and Nanjing University.

### HPI RESEARCH GROUPS

- **Algorithm Engineering**, Prof. Dr. Tobias Friedrich
- **Business Process Technology**, Prof. Dr. Mathias Weske
- **Computer Graphics Systems**, Prof. Dr. Jürgen Döllner
- **Enterprise Platform and Integration Concepts**, Prof. Dr. h.c. Hasso Plattner
- **Human Computer Interaction**, Prof. Dr. Patrick Baudisch
- **Information Systems**, Prof. Dr. Felix Naumann
- **Internet Technologies and Systems**, Prof. Dr. Christoph Meinel
- **Knowledge Discovery and Data Mining**, Prof. Dr. Emmanuel Müller
- **Operating Systems and Middleware**, Prof. Dr. Andreas Polze
- **Software Architecture**, Prof. Dr. Robert Hirschfeld
- **System Engineering and Modeling**, Prof. Dr. Holger Giese

Applications must be submitted by August 15 of the respective year.  
For more information on the HPI Research School please visit:  
[www.hpi.de/research-school](http://www.hpi.de/research-school)



## ADVERTISING IN CAREER OPPORTUNITIES

**How to Submit a Classified Line Ad:**  
**Send an e-mail to [acmm mediasales@acm.org](mailto:acmm mediasales@acm.org).** Please include text, and indicate the issue/or issues where the ad will appear, and a contact name and number.

**Estimates:** An insertion order will then be e-mailed back to you. The ad will be typeset according to CACM guidelines. **NO PROOFS** can be sent. Classified line ads are **NOT** commissionable.

**Rates:** \$325.00 for six lines of text, 40 characters per line. \$32.50 for each additional line after the first six. The **MINIMUM** is six lines.

**Deadlines:** 20th of the month/ 2 months prior to issue date.  
**For latest deadline info, please contact:**

[acmm mediasales@acm.org](mailto:acmm mediasales@acm.org)

**Career Opportunities Online:**  
Classified and recruitment display ads receive a free duplicate listing on our website at:

<http://jobs.acm.org>

**Ads are listed for a period of 30 days.**

**For More Information Contact:**

**ACM Media Sales,  
at 212-626-0686 or  
[acmm mediasales@acm.org](mailto:acmm mediasales@acm.org)**



上海科技大学  
ShanghaiTech University

### TENURE-TRACK AND TENURED POSITIONS IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

The newly launched ShanghaiTech University is built as a world-class research university, which locates in Zhangjiang High-Tech Park. We invite highly qualified candidates to fill tenure-track/tenured faculty positions as its core team in the School of Information Science and Technology (SIST). Candidates should have exceptional academic records or demonstrate strong potential in cutting-edge research areas of information science and technology. They must be fluent in English. Overseas academic connection or background is highly desired.

#### Academic Disciplines:

We seek candidates in all cutting edge areas of information science and technology. Our recruitment focus includes, but is not limited to: computer architecture and technologies, nano-scale electronics, high speed and RF circuits, intelligent and integrated signal processing systems, computational foundations, big data, data mining, visualization, computer vision, bio-computing, smart energy/power devices and systems, next-generation networking, as well as inter-disciplinary areas involving information science and technology.

#### Compensation and Benefits:

Salary and startup funds are highly competitive, commensurate with experience and academic accomplishment. We also offer a comprehensive benefit package to employees and eligible dependents, including housing benefits. All regular ShanghaiTech faculty members will be within its new tenure-track system commensurate with international practice for performance evaluation and promotion.

#### Qualifications:

- A detailed research plan and demonstrated record/potentials;
- Ph.D. (Electrical Engineering, Computer Engineering, Computer Science, or related field);
- A minimum relevant research experience of 4 years.

#### Applications:

Submit (in English, PDF version) a cover letter, a 2-page research plan, a CV plus copies of 3 most significant publications, and names of three referees to: [sist@shanghaitech.edu.cn](mailto:sist@shanghaitech.edu.cn) (until positions are filled). For more information, please visit Job Opportunities on <http://sist.shanghaitech.edu.cn/>

**Deadline: July 31, 2016**



### Associate/Full Professor of Cyber Security

The cyber security section of the Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS) focuses on network security, secure data processing, and situation awareness in cyberspace. The section's research and education portfolio is rich and deals with several perspectives on cyber security. We value the diversity of our portfolio within the scope of theory and engineering of cyber security for distributed systems and networks in a socio-technical context.

#### Level:

PhD degree

#### Maximum employment:

38 hours per week (1 FTE)

#### Duration of contract:

Fixed appointment

#### Salary scale:

€ 5219 to € 7599 per month gross

#### Contact

Prof. R. L. Lagendijk

+31 (0)15-2783731

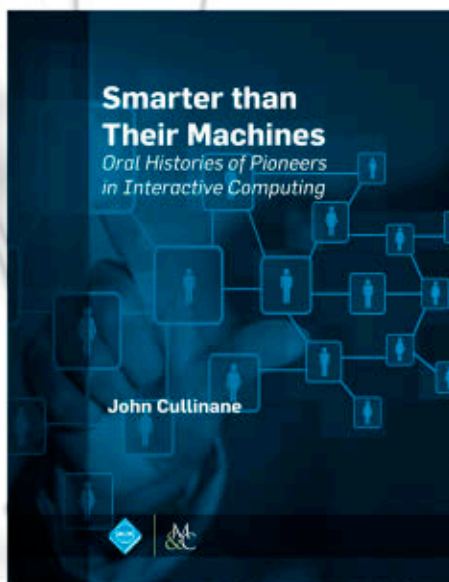
[R.L.Lagendijk@tudelft.nl](mailto:R.L.Lagendijk@tudelft.nl)

For more information and the requirements, visit:

[www.jobsindelft.com](http://www.jobsindelft.com) or

<http://cys.ewi.tudelft.nl/cys/vacancies>

The Assoc./Full Professor of cyber security will function as the mainstay for cyber security research and education, without assuming that (s)he is expert in all aspects of the section's scope. (S)he will be employed at the EEMCS Faculty and will continue the commenced strategy to leverage the complementary expertise at the Faculties of EEMCS and Technology, Policy and Management (TPM). The aim is the creation of joint research projects as well as education and technology transfer activities, thus combining the computer science and socio-technical perspective on cyber security.



## A personal walk down the computer industry road. BY AN EYEWITNESS.

**Smarter Than Their Machines: Oral Histories of the Pioneers of Interactive Computing** is based on oral histories archived at the Charles Babbage Institute, University of Minnesota. These oral histories contain important messages for our leaders of today, at all levels, including that government, industry, and academia can accomplish great things when working together in an effective way.



ISBN: 978-1-62705-550-5 DOI: 110.1145/2663015

<http://books.acm.org>

<http://www.morganclaypoolpublishers.com/acm>



# Upstart Puzzles

## Chair Games

A GROUP OF people is sitting around your dinner table with one empty chair. Each person has a name that begins with a different letter: A, B, C ... Because you love puzzles, you ask them to rearrange themselves to end up in alphabetical order in a clockwise fashion, with one empty chair just to the left of the person whose name begins with A. Each move involves one person from one chair to the empty chair  $k$  seats away in either direction. The goal is to minimize the number of such moves.

*Warm-up.* Suppose you start with eight people around the table, with nine chairs. The last name of each person begins with the letter shown, and you are allowed to move a person from one chair to an empty chair three chairs away in either direction (see Figure 1). Can you do it in four moves?

*Solution to Warm-Up*  
C moves from 9 to 6  
F moves from 3 to 9  
C moves from 6 to 3  
F moves from 9 to 6  
Now here is a more difficult version of the problem in which the only moves allowed are four seats away, starting with the configuration in Figure 2.

*Solution*  
B moves from 6 to 2  
F moves from 1 to 6  
A moves from 5 to 1  
E moves from 9 to 5  
Now try to solve these four related upstart puzzles:

*Number of people  $n$  and move distance  $k$ .* Given any number of people  $n$  and move distance  $k$ , find the minimum number of moves that would create a clockwise sorted order;

*Number of people  $n$  in a certain arrangement, find best move distance.* Given any number of people  $n$ , find a move distance  $k$  and the minimum number of moves of length  $k$  that creates a clockwise sorted order;

*Number  $n$  of people, one empty chair.* Generalize the problem with  $n$  people and one empty chair to allow movements of distances  $k_1, k_2, \dots, k_j$  for some  $j < n/2$ ; and

*Several empty chairs.* Generalize the problem further to allow several empty chairs. C

All are invited to submit solutions and prospective upstart-style puzzles for future columns to [upstartpuzzles@cacm.acm.org](mailto:upstartpuzzles@cacm.acm.org)

**Dennis Shasha** ([dennisshasha@yahoo.com](mailto:dennisshasha@yahoo.com)) is a professor of computer science in the Computer Science Department of the Courant Institute at New York University, New York, as well as the chronicler of his good friend the omniheurist Dr. Ecco.  
Copyright held by author.

Figure 1 (warm-up). The goal is to rearrange the people around the table to be in clockwise alphabetical order, using as few moves as possible, where each move involves moving a person three seats away (in either direction) from the empty chair.



Figure 2. The goal is again to rearrange the people around the table to be in clockwise alphabetical order, using as few moves as possible, where each move involves moving a person four seats away (in either direction) from the empty chair.



# Computing Reviews

## Connect with our Community of Reviewers

---

*“There aren’t too many walk-in technical bookshops where I live; checking out reviewer comments on new books provides me with an equivalent experience.”*

- Graham Jenkins

---



Association for  
Computing Machinery

ThinkLoud

[www.computingreviews.com](http://www.computingreviews.com)



ACM Symposium on  
Spatial User Interaction  
Tokyo, Japan  
15-16 October 2016  
Co-located with ACM UIST

Sponsored by



<http://sui-symposium.org>

Welcome to **SUI 2016** (rhymes with GUI, スウイ), is a new and upcoming venue for the dissemination of future user interfaces. We invite you to enjoy presentations of leading-edge research results in a comfortable atmosphere. ***Selected SUI papers and demos will be presented in a special poster/demo track at UIST 2016.***

#### Deadlines

Paper Submissions: 1 July 2016

Poster and Demo Submissions: 9 August 2016

The Future of User Interfaces  
「未来ユーザインタフェース」



Keynote Speaker:  
**Shahram Izadi**  
(Microsoft Research)



General Chair:  
**Christian Sandor**  
(Nara Institute of  
Science and Technology)



Japanese Hospitality & Tradition - おもてなし

Gozabune

Sumie



#### Panelists:

**Otmar Hilliges**  
(ETH Zürich)

**Hrvoje Benko**  
(Microsoft Research)

**Alex Olwal**  
(Google Inc.)

#### Moderator:

**Aitor Rovira**  
(Nara Institute of  
Science and Technology)