

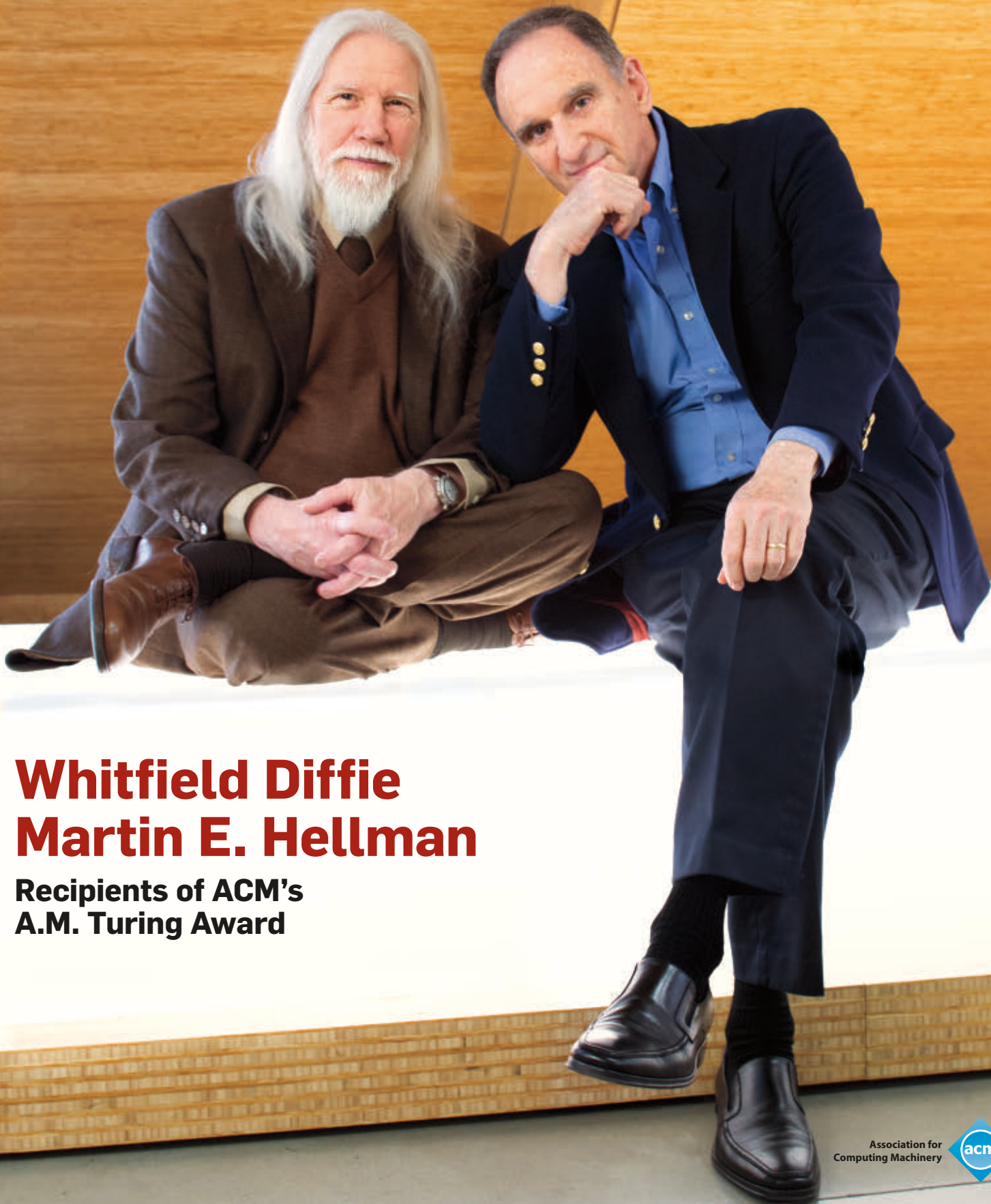
COMMUNICATIONS

CACM.ACM.ORG

OF THE

ACM

06/2016 VOL.59 NO.06

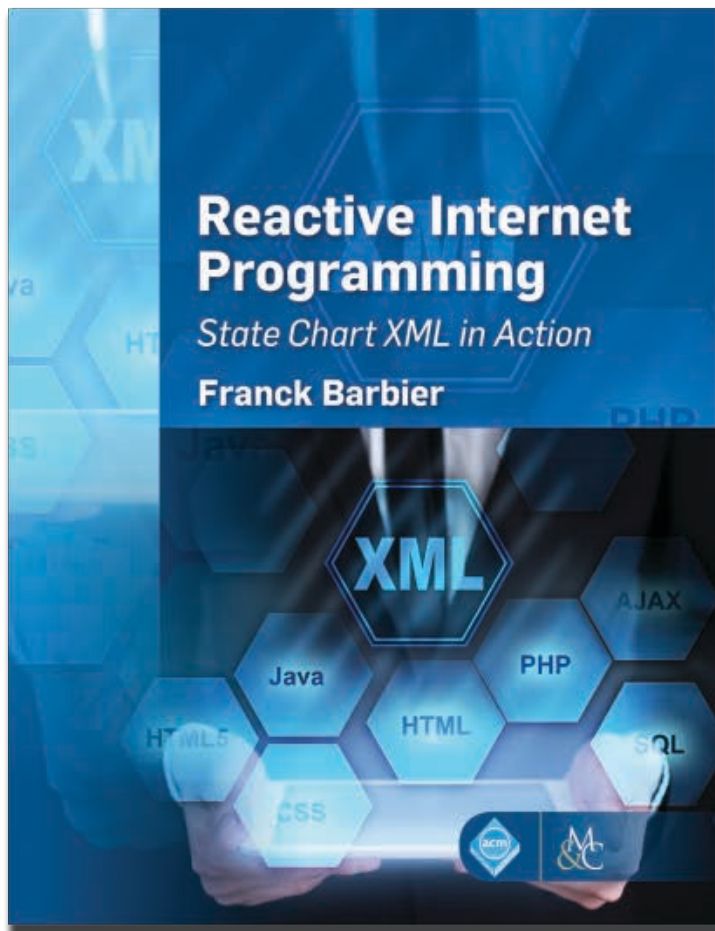


Whitfield Diffie Martin E. Hellman

Recipients of ACM's
A.M. Turing Award

Association for
Computing Machinery





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>

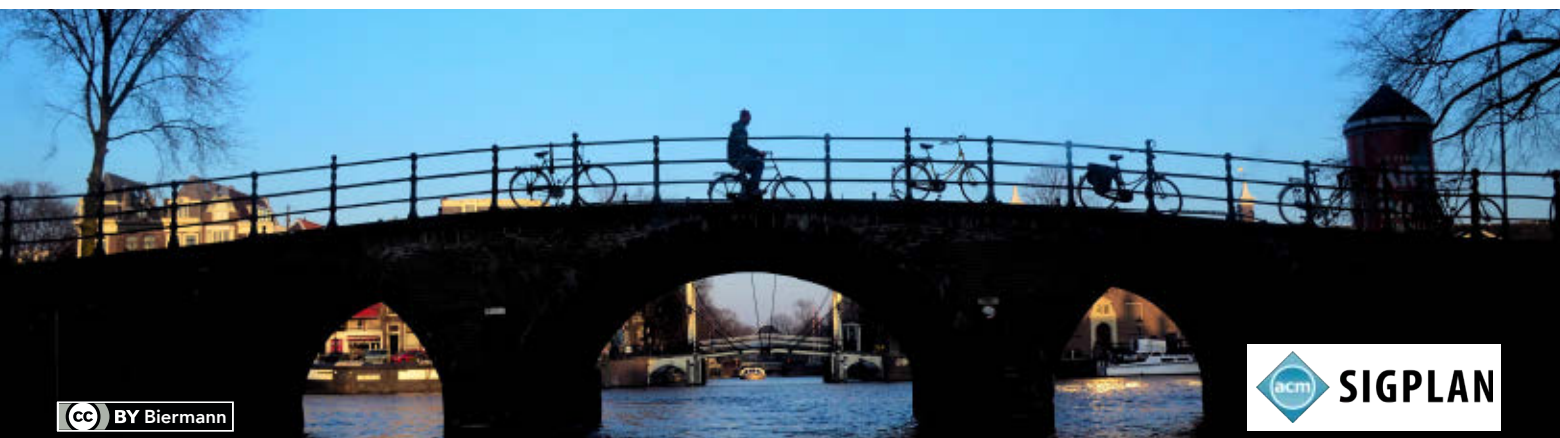
this.splash 2016

Amsterdam

Sun 30 October – Fri 4 November 2016

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

- OOPSLA** Novel research on software development and programming
- Onward!** Radical new ideas and visions related to programming and software
- SPLASH-I** World class speakers on current topics in software, systems, and languages research
- SPLASH-E** Researchers and educators share educational results, ideas, and challenges
 - DLS** Dynamic languages, implementations, and applications
 - GPCE** Generative programming: concepts and experiences
 - SLE** 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

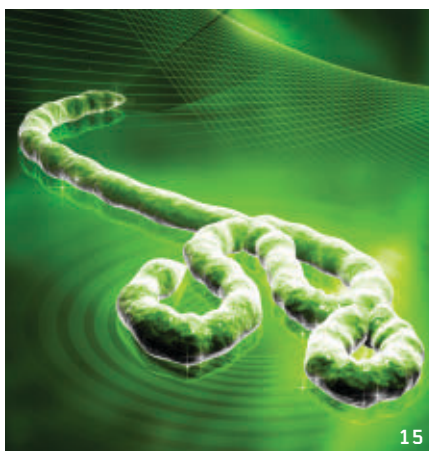
Departments

- 5 **From the President**
Moving Forward
By Alexander L. Wolf
-
- 7 **Cerf's Up**
Celebrations!
By Vinton G. Cerf
-
- 8 **Letters to the Editor**
No Backdoor Required or Expected
-
- 10 **BLOG@CACM**
The Solution to AI, What Real Researchers Do, and Expectations for CS Classrooms
John Langford on AlphaGo, Bertrand Meyer on Research as Research, and Mark Guzdial on correlating CS classes with laboratory results.
-
- 29 **Calendar**

Last Byte

- 112 **Q&A**
Finding New Directions in Cryptography
Whitfield Diffie and Martin Hellman on their meeting, their research, and the results that billions use every day.
By Leah Hoffmann

News



15

- 12 **Turing Profile**
The Key to Privacy
40 years ago, Whitfield Diffie and Martin Hellman introduced the public key cryptography used to secure today's online transactions.
By Neil Savage



Watch the Turing recipients discuss their work in this exclusive *Communications* video.
<http://cacm.acm.org/videos/the-key-to-privacy>

- 15 **What Happens When Big Data Blunders?**
Big data is touted as a cure-all for challenges in business, government, and healthcare, but as disease outbreak predictions show, big data often fails.
By Logan Kugler

- 17 **Reimagining Search**
Search engine developers are moving beyond the problem of document analysis, toward the elusive goal of figuring out what people really want.
By Alex Wright

- 20 **What's Next for Digital Humanities?**
New computational tools spur advances in an evolving field.
By Gregory Mone

Viewpoints



26

- 22 **Inside Risks**
The Risks of Self-Auditing Systems
Unforeseen problems can result from the absence of impartial independent evaluations.
By Rebecca T. Mercuri and Peter G. Neumann

- 26 **Kode Vicious**
What Are You Trying to Pull?
A single cache miss is more expensive than many instructions.
By George V. Neville-Neil

- 28 **The Profession of IT**
How to Produce Innovations
Making innovations happen is surprisingly easy, satisfying, and rewarding if you start small and build up.
By Peter J. Denning

- 31 **Interview**
An Interview with Yale Patt
ACM Fellow Professor Yale Patt reflects on his career.
By Derek Chiou



Watch Patt discuss his work in this exclusive *Communications* video.
<http://cacm.acm.org/videos/an-interview-with-yale-patt>

For the full-length video, please visit <https://vimeo.com/an-interview-with-yale-patt>



Association for Computing Machinery
Advancing Computing as a Science & Profession

Viewpoints (cont'd.)

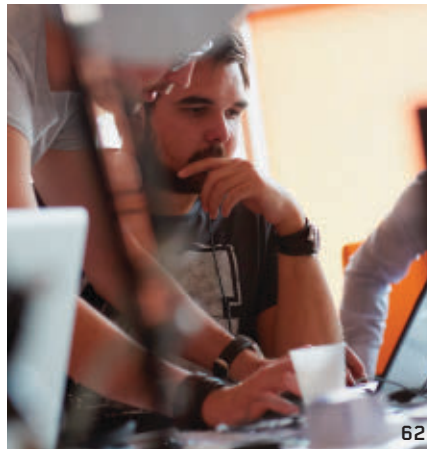
- 37 **Viewpoint**
Computer Science Should Stay Young
Seeking to improve computer science publication culture while retaining the best aspects of the conference and journal publication processes.
By Boaz Barak
-
- 39 **Viewpoint**
Privacy Is Dead, Long Live Privacy
Protecting social norms as confidentiality wanes.
By Jean-Pierre Hubaux and Ari Juels
-
- 42 **Viewpoint**
A Byte Is All We Need
A teenager explores ways to attract girls into the magical world of computer science.
By Ankita Mitra

Practice

- 45 **Nine Things I Didn't Know I Would Learn Being an Engineer Manager**
Many of the skills aren't technical at all.
By Kate Matsudaira
-
- 48 **The Flame Graph**
This visualization of software execution is a new necessity for performance profiling and debugging.
By Brendan Gregg
-
- 58 **Standing on Distributed Shoulders of Giants**
Farsighted physicists of yore were danged smart!
By Pat Helland

Q Articles' development led by **acmqueue**
queue.acm.org

Contributed Articles



62

- 62 **Improving API Usability**
Human-centered design can make application programming interfaces easier for developers to use.
By Brad A. Myers and Jeffrey Stylos
-
- 70 **Physical Key Extraction Attacks on PCs**
Computers broadcast their secrets via inadvertent physical emanations that are easily measured and exploited.
By Daniel Genkin, Lev Pachmanov, Itamar Pipman, Adi Shamir, and Eran Tromer



About the Cover:
Whitfield Diffie (left) and Martin E. Hellman, cryptography pioneers and recipients of the 2015 ACM A.M. Turing Award, photographed at Stanford University's Huang Center in March. Photographed by Richard Morgenstein, <http://www.morgenstein.com/>

Review Articles



80

- 80 **RandNLA: Randomized Numerical Linear Algebra**
Randomization offers new benefits for large-scale linear computations.
By Petros Drineas and Michael W. Mahoney

Research Highlights

- 92 **Technical Perspective**
Veritestng Tackles Path-Explosion Problem
By Koushik Sen
-
- 93 **Enhancing Symbolic Execution with Veritestng**
By Thanassis Avgerinos, Alexandre Rebert, Sang Kil Cha, and David Brumley
-
- 101 **Technical Perspective**
Computing with the Crowd
By Siddharth Suri
-
- 102 **AutoMan: A Platform for Integrating Human-Based and Digital Computation**
By Daniel W. Barowy, Charlie Curtsinger, Emery D. Berger, and Andrew McGregor



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-Neil

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.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

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

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-Neil; Theo Schlossnagle;
Jim Waldo

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

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 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.

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.

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:10.1145/2933245

Alexander L. Wolf

Moving Forward

AS MY TENURE as ACM president ends, I find myself reflecting on the past two years and so I looked back at my 2014 election position statement.

[W]e must confront the reality that what ACM contributed to the computing profession for more than 65 years might not sustain it in the future ...

ACM was formed in 1947 by a small group of scientists and engineers who had helped usher in the computer age during World War II. They saw ACM as a means for professionals, primarily mathematicians and electrical engineers, to exchange and curate technical information about “computing machinery.” The fact that ACM is now home to a much broader community of interests, with members literally spanning the globe, was likely well beyond their imagination.

Conferences and publications remain the primary means by which our organization sustains itself. I worried in 2014 that revenue would eventually fall, and that we needed to prepare. I pointed out in a 2015 *Communications* letter that conference surpluses go directly back to the SIGs, while publication surpluses are used to subsidize the entire enterprise: allowing student members everywhere, and reduced-rate professional members in developing regions, to receive full member benefits; contributing an additional \$3M per year to the SIGs; and supporting in entirety our volunteer-driven efforts in education, inclusion, and public policy. The specter of open access undercutting the library subscription business created many uncertainties, some of which remain to this day.

Two years on, some things are coming into better focus, giving hope that conferences and publications will remain viable revenue sources.

As it turns out, the popularity of our conferences continues to rise with overall conference attendance steadily increasing. I attribute this to the growing importance and influence of computing, and the broadening of ACM’s constituency and audience.

We have empowered authors and conference organizers with new open access options. Yet the uptake of Gold (“author pays”) open access is surprisingly slow and the growth of the subscription business is surprisingly robust. Perhaps most profound is the realization that the marketable value of ACM’s Digital Library derives not so much from access to individual articles, as from access to the *collection* and the *services* that leverage and enhance the collection. In other words, ACM sells subscriptions to a collection, so in a sense open access to articles is not the immediate threat. Moreover, there is a potential future business to be built around government mandates for *open data*, *reproducible computation*, and *digital preservation* generally that takes us far beyond today’s simple PDF artifact and collection index.

We must recognize that the nature of community, community identity, and “belonging” is evolving rapidly ...

What is the value of being formally associated with ACM? This seemingly simple and fundamental question comes up so often that the answer should be obvious and immediate. Twenty years ago, perhaps it was. Today, although I personally feel the value, I struggle to articulate an answer that I am confident will convince someone new to the community already engaged with others through means falling outside the traditional ACM circle.

What I do know is that remarkably few people are aware of the important and impactful volunteer activities beyond conferences and publications

that are supported by ACM. This seems to be the case whether the person is one of our more than 100,000 dues-paying members or one of the millions of non-dues-paying participants and beneficiaries in ACM activities.

That is why I sought to “change the conversation” around ACM, from merely serving as computing’s premier conference sponsor and publisher to also being a potent and prominent force for good in the community. My goal was to raise awareness that ACM, as a *professional* society, offers a uniquely authoritative, respected voice, one that can amplify the efforts of individuals in a way that an ad hoc social network cannot. That ACM and its assets are at the disposal of its members and volunteer leaders to drive its agenda forward. And that being a *member* of this organization is a statement in support of that agenda. Getting this message out is largely about how ACM presents itself to the world through its communication channels, which are in the process of a long-overdue refresh.

ACM’s services and programs are founded on three vital pillars: energetic volunteers, dedicated HQ staff, and a sufficient and reliable revenue stream ...

The most rewarding experiences I had as president were visits with the many communities within the community that is ACM: SIGs, chapters, boards, and committees. Each different, yet bound by a commitment to excellence that is our organization’s hallmark. Enabling those communities is a professional staff as passionate about ACM as its members. They deserve our thanks and respect.

As I end my term, I wish the next president well in continuing to move the organization forward. You have great people to work with and an important legacy to continue. 

Alexander L. Wolf is president of ACM and a professor in the Department of Computing at Imperial College London, UK.

Copyright held by author.

The National Academies of
SCIENCES • ENGINEERING • MEDICINE

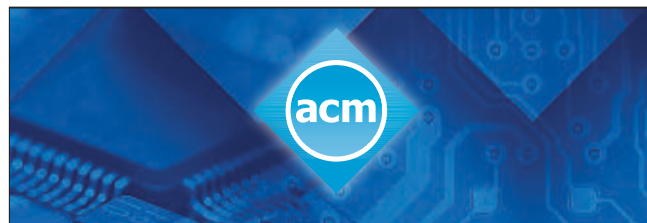
ARL Distinguished Postdoctoral Fellowships

The Army Research Laboratory (ARL) is the nation's premier laboratory for land forces. The civilians working at ARL and its predecessors have had many successes in basic and applied research. Currently, ARL scientists and engineers are pioneering research in such areas as neuroscience, energetic materials and propulsion, electronics technologies, network sciences, virtual interfaces and synthetic environments and autonomous systems. They are leaders in modeling and simulation and have high performance computing resources on-site. They are expanding into frontier areas, including fields such as quantum information and quantum networks.

We invite outstanding young researchers to participate in this excitement as ARL Distinguished Postdoctoral Fellows. These Fellows will display extraordinary ability in scientific research and show clear promise of becoming outstanding future leaders. Candidates are expected to have already successfully tackled a major scientific or engineering problem or to have provided a new approach or insight evidenced by a recognized impact in their field. ARL offers these named Fellowships in honor of distinguished researchers and work that has been performed at Army labs.

The ARL Distinguished Postdoctoral Fellowships are three-year appointments. The annual stipend is \$100,000, and the fellowship includes benefits and potential additional funding for selected proposals. Applicants must hold a Ph.D., awarded within the past three years, at the time of application. For complete application instructions and more information, visit: <http://sites.nationalacademies.org/PGA/Fellowships/ARL>.

Applications must be received by July 1, 2016.



Advertise with ACM!

Reach the innovators and thought leaders
working at the cutting edge of
computing and information
technology through ACM's magazines,
websites and newsletters.

Request a media kit
with specifications and pricing:



Craig Pitcher

408-778-0300 ♦ pitcherc@ACM.org

Bill Sleight

408-513-3408 ♦ wsleight@ACM.org

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.

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
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



DOI:10.1145/2933148

Vinton G. Cerf

Celebrations!

There is a rhythm in the affairs of the Association for Computing Machinery and June marks our annual celebration of award recipients and the biennial election of new

officers. I will end my final year as past president, Alex Wolf will begin his first year in that role, and a new president and other officers will take their places in the leadership. June also marks Bobby Schnabel's first appearance at our annual awards event in his role as CEO of ACM. I am especially pleased that two former Stanford colleagues, Martin Hellman and Whitfield Diffie, are receiving the ACM A.M. Turing Award this year. Nearly four decades have passed since their seminal description of what has become known as public key cryptography and in that time the technology has evolved and suffused into much of our online and offline lives.

In another notable celebration, Alphabet, the holding company that includes Google, saw its AlphaGo system from DeepMind win four of five GO games in Seoul against a world class human player. The complexity of the state space of GO far exceeds that of chess and many of us were surprised to see how far neural networks have evolved in what seems such a short period of time. Interestingly, the system tries to keep track of its own confidence level as it uses the state of the board to guide its choices of next possible moves. We are reminded once again how complexity arises from what seems to be the simplest of rules.

While we are celebrating advances in artificial intelligence, other voices are forecasting a dark fate for humanity. Intelligent machines, once they can match a human capacity, will go

on to exceed it, they say. Indeed, our supercomputers and cloud-based systems can do things that no human can do, particularly with regard to "big data." Some of us, however, see the evolution of computing capability in terms of partnership. When you do a search on the World Wide Web or use Google to translate from one language to another, you are making use of powerful statistical methods, parsing, and semantic graphs to approximate what an accomplished multilingual speaker might do. These translations are not perfect but they have been improving over time. This does not mean, however, that the programs *understand* in the deepest cognitive sense what the words and sentences mean. In large measure, such translation rests on strong correlation and grammar. This is not to minimize the utility of such programs—they enhance our ability to communicate across language barriers. They can also create confusion when misinterpretation of colloquialisms or other nuances interfere with precision.

One has to appreciate, however, the role of robotics in manufacturing in today's world. The Tesla factory in Fremont, CA, is a marvel of automation^a and there are many other examples, including the process of computer chip production that figures so strongly in the work of ACM's members. Automation can be considered an aspect of artificial in-

telligence if by this we mean the autonomous manipulation of the real world. Of course, one can also argue, as I have in the past, that stock market trading programs are robotic in the sense they receive inputs, perform analysis, and take actions that affect the real world (for example, our bank accounts). Increasingly, we see software carrying out tasks in largely autonomous ways, including the dramatic progress made in self-driving cars. Apart from what we usually call artificial intelligence, it seems important to think about software that goes about its operation with little or no human intervention. I must confess, I am still leery of the software that runs the massage chairs at Google—thinking that a bug might cause the chair to fold up while I am sitting in it!

While we celebrate the advances made in artificial intelligence and autonomous systems, we also have an obligation to think deeply about potential malfunctions and their consequences. This certainly persuades me to keep in mind safety and reliability to say nothing of security, privacy, and usability, as we imbue more and more appliances and devices with programmable features and the ability to communicate through the Internet. **□**

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

^a <https://www.youtube.com/watch?v=TuC8drQmXjg>

Copyright held by author.

DOI:10.1145/2931085

No Backdoor Required or Expected

I WAS DISAPPOINTED by Eugene H. Spafford's column "The Strength of Encryption" (Mar. 2016) in which Spafford conflated law enforcement requests for access to the contents of specific smartphones with the prospect of the government requiring backdoors through which any device could be penetrated. These are separate issues. Even if the methods the FBI ultimately used to unlock a particular Apple iPhone 5C earlier this year are too elaborate for the hundreds of encrypted or code-protected phones now in police custody, the principle—that it is a moral if not legal responsibility for those with the competence to open the phones do so—would still be relevant.

Unlocking an individual phone would not legally compel a backdoor into all Apple devices. Rather, Apple would have to create and download into a particular target phone only a version of iOS that does two things—return to requesting password entry after a failed attempt, without invoking the standard iOS delay-and-attempt-count code and allow password attempts at guessing the correct password be submitted electronically rather than through physical taps on the phone's keypad. The first is clearly trivial, and the second is, I expect, easily achieved.

The FBI would then observe, at an Apple facility, the modified iOS being downloaded and be able to run multiple brute-force password attempts against it. When the phone is eventually unlocked, the FBI would have the former user's correct password. Apple could then reload the original iOS, and the FBI could take away the phone and the password and access the phone's contents without further Apple involvement.

No backdoor would have been released. No existing encryption security would have been compromised. Other law-enforcement agencies, armed with

judicial orders, would likewise expect compliance—and should receive it.

The secondary argument—that should Apple comply and authoritarian regimes worldwide would demand the same sort of compliance from Apple, as well as from other manufacturers—is a straw man. Since Apple and other manufacturers, as well as researchers, have acknowledged they are able to gain access to the contents of encrypted phones, other regimes are already able to make such demands, independent of the outcome of any specific case.

R. Gary Marquart, Austin, TX

Author Responds:

My column was written and published before the FBI vs. Apple lawsuit occurred and was on the general issue of encryption strength and backdoors. Nowhere in it did I mention either Apple or the FBI. I also made no mention of "unlocking" cellphones, iOS, or passwords. I am thus unable to provide any reasonable response to Marquart's objections as to items not in it.

Eugene H. Spafford, West Lafayette, IN

The What in the GNU/Linux Name

George V. Neville-Neil's Kode Vicious column "GNL Is Not Linux" (Apr. 2016) would have been better if it had ended with the opening paragraph. Instead Neville-Neil recapped yet again the history of Unix and Linux, then went off the rails, hinting, darkly, at ulterior motives behind GPL, particularly that it is anti-commercial. Red Hat's billions in revenue (\$1.79 billion in 2015) should put such an assertion to rest. The Free Software Foundation apparently has no problem with individuals or companies making money from free software.

We do not call houses by the tools we use to build them, as in, say, "... a Craftsman/House, a Makita/House, or a Home Depot/House ..." in Neville-Neil's example. But we do call a house

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

made of bricks a brick house in a nomenclature that causes no confusion. Why then would it be confusing to call a system with a Linux kernel and a user space largely from the GNU project a “GNU/Linux system”? Including “GNU” in the name seems to be a problem only for people with an anti-GNU bias or misunderstanding of GPL, both of which Neville-Neil exhibited through his “supposedly” slight (in paragraph 10) intended to cast aspersions on the Hurd operating system project and the dig (as I read it) at GPLv3 for being more restrictive than GPLv2. However, in fairness, GPLv3 is more restrictive and explicit about not allowing patents to circumvent the freedoms inherent in a license otherwise granted by copyright. As Neville-Neil appeared disdainful of the GPLv2 methods of securing users’ freedoms, it is not surprising he would take a negative view of GPLv3.

Neville-Neil also suggested the “GNU/Linux” name is inappropriate, as it reflects the tools used to build the kernel. But as Richard Stallman explained in his 2008 article “Linux and the GNU System” (<http://www.gnu.org/gnu/linux-and-gnu.html>) to which Neville-Neil linked in his column, a typical Linux distribution includes more code from the GNU project than from the Linux kernel project. Perhaps Neville-Neil should pour himself a less-“strong beverage” and read Stallman’s article again. He may find himself much less confused by the “GNU/Linux” name.

Todd M. Lewis, Sanford, NC

Author Responds:

Lewis hints at my anti-GPL bias, though I have been quite direct in my opposition to any open source license that restricts the freedoms of those using the code, as is done explicitly by the GPLv2 licenses. Open source means just that—open, free to everyone, without strings, caveats, codicils, or clawbacks. As for a strong drink and a re-read of anything from Richard Stallman it would have to be a very strong drink indeed to induce me to do it again.

George V. Neville-Neil, Brooklyn, NY

Diversity and ‘CS for All’

Vinton G. Cerf’s Cerf’s Up column “Enrollments Explode! But diversity students are leaving ...” (Apr. 2016) on di-

versity in computer science education and Lawrence M. Fisher’s news story on President Barack Obama’s “Computer Science for All” initiative made us think *Communications* readers might be interested in our experience at Princeton University over the past decade dramatically increasing both CS enrollments in general and the percentage of women in CS courses. As of the 2015–2016 academic year, our introductory CS class was the highest-enrolled class at Princeton and included over 40% women, with the number and percentage of women CS majors approaching similar levels.

Our approach is to teach a CS course for everyone, focusing outwardly on applications in other disciplines, from biology and physics to art and music.¹ We begin with a substantive programming component, with each concept introduced in the context of an engaging application, ranging from simulating the vibration of a guitar string to generate sound to implementing Markov language models to computing DNA sequence alignments. This foundation allows us to consider the great intellectual contributions of Turing, Shannon, von Neumann, and others in a scientific context. We have also had success embracing technology, moving to active learning with online lectures.² We feel CS is something every college student can and must learn, no matter what their intended major, and there is much more to it than programming alone. Weaving CS into the fabric of modern life and a broad educational experience in this way is valuable to all students, particularly women and underrepresented minorities. Other institutions adopting a similar approach have had similar success.

Meanwhile, we have finally (after 25 years of development) completed our CS textbook *Computer Science, An Interdisciplinary Approach* (Addison-Wesley, 2016), which we feel can stand alongside standard textbooks in biology, physics, economics, and other disciplines. It will be available along with studio-produced lectures and associated Web content (<http://introcs.cs.princeton.edu>) that attract more than one million visitors per year.

Over the next few years, we will seek opportunities to disseminate these

materials to as many teachers and learners as possible. Other institutions will be challenged to match our numbers, particularly percentage of women engaged in CS. It is an exciting time.

References

1. Hulett, D. ‘Computer Science for All’ (Really). Princeton University, Princeton, NJ, Mar. 1, 2016; <https://www.cs.princeton.edu/news/2016/03/01/computer-science-all-really>
2. Sedgewick, R. *A 21st Century Model for Disseminating Knowledge*. Princeton University, Princeton, NJ; <http://www.cs.princeton.edu/~rs/talks/Model.pdf>

Robert Sedgewick and Kevin Wayne, Princeton, NJ

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.

© 2016 ACM 0001-0782/16/06 \$15.00

Coming Next Month in **COMMUNICATIONS**

The Rise of Social Bots

Statistics for Engineers

On the Growth of Polyominoes

Turing’s Red Flag

Should You Upload or Ship Big Data to the Cloud?

Inverse Privacy

Formula-Based Software Debugging

The Motivation for a Monolithic Codebase

Mesa: Geo-Replicated Online Data Warehouse for Google’s Advertising System

Plus the latest news about solving graph isomorphism, AI and the LHC, and apps that fight parking tickets.

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:10.1145/2911969

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

The Solution to AI, What Real Researchers Do, and Expectations for CS Classrooms

John Langford on AlphaGo, Bertrand Meyer on Research as Research, and Mark Guzdial on correlating CS classes with laboratory results.



John Langford
AlphaGo Is Not the Solution to AI
<http://bit.ly/1QSqgHW>
March 14, 2016

Congratulations are in order for the folks at Google Deepmind (<https://deepmind.com>) who have mastered Go (<https://deepmind.com/alpha-go.html>).

However, some of the discussion around this seems like giddy overstatement. *Wired* says, “machines have conquered the last games” (<http://bit.ly/200O5zG>) and *Slashdot* says, “we know now that we don’t need any big new breakthroughs to get to true AI” (<http://bit.ly/1q0Pcmg>). The truth is nowhere close.

For Go itself, it has been well known for a decade that Monte Carlo tree search (MCTS, <http://bit.ly/1YbLm4M>; that is, valuation by assuming randomized playout) is unusually effective in Go. Given this, it is unclear the AlphaGo algorithm extends to other board games

where MCTS does not work so well. Maybe? It will be interesting to see.

Delving into existing computer games, the Atari results (<http://bit.ly/1YbLBgl>, Figure 3) are very fun but obviously unimpressive on about a quarter of the games. My hypothesis for why their solution does only local (epsilon-greedy style) exploration rather than global exploration so they can only learn policies addressing either very short credit assignment problems or with greedily accessible policies. Global exploration strategies are known to result in exponentially more efficient strategies in general for deterministic decision process (1993, <http://bit.ly/1YbLKjQ>), Markov Decision Processes (1998, <http://bit.ly/1RXTRCk>), and for MDPs without modeling (2006, <http://bit.ly/226J1tc>).

The reason these strategies are not used is because they are based on tabular learning rather than function fitting. That is why I shifted to Contextual Bandit research (<http://bit.ly/1S4iiHT>) after the 2006 paper. We have learned quite a

bit there, enough to start tackling a Contextual Deterministic Decision Process (<http://arxiv.org/abs/1602.02722>), but that solution is still far from practical. Addressing global exploration effectively is only one of the significant challenges between what is well known now and what needs to be addressed for what I would consider a real AI.

This is generally understood by people working on these techniques but seems to be getting lost in translation to public news reports. That is dangerous because it leads to disappointment (<http://bit.ly/1q11dDW>). The field will be better off without an overpromise/bust cycle, so I would encourage people to keep and inform a balanced view of successes and their extent. Mastering Go is a great accomplishment, but it is quite far from everything.

See further discussion at <http://bit.ly/20106Ff>.



Bertrand Meyer
What's Your Research?
<http://bit.ly/1QRo9Q9>
March 3, 2016

One of the pleasures of having a research activity is that you get to visit research institutions and ask people what they do. Typically, the answer is “I work in X” or “I work in the application of X to Y,” as in (made-up example among countless ones, there are many Xs and many Ys): *I work in model checking for distributed systems*. Notice the “in.”

This is, in my experience, the dominant style of answers to such a question. I find it disturbing. It is about research as a job, not research as research.

Research is indeed, for most researchers, a job. It was not always like that: up to the time when research took on its modern form, in the 18th and early 19th centuries, researchers were people employed at something else, or fortunate enough not to need employment, who spent some of their time looking into open problems of science. Now research is something that almost all its practitioners do for a living.

But a real researcher does not just follow the flow, working “in” a certain fashionable area or at the confluence of two fashionable areas. A real researcher attempts to solve open problems.

This is the kind of answer I would expect: I am trying to find a way to do A, which no one has been able to do yet; or to find a better way to do B, because the current ways are deficient; or to solve the C conjecture as posed by M; or to find out why phenomenon D is happening; or to build a tool that will address need E.

A researcher does not work “in” an area but “on” a question.

This observation also defines what it means for research to be successful. If you are just working “in” an area, the only criteria are bureaucratic: paper accepted, grant obtained. They cover the means, not the end. If you view research as problem solving, success is clearly and objectively testable: you solved the problem you set out to solve, or not. Maybe that is the reason we are uneasy with this view: it prevents us from taking cover behind artificial and deceptive proxies for success.

Research is about solving problems; at least about trying to solve a problem, or—more realistically and modestly—bringing your own little incremental contribution to the ongoing quest for a solution. We know our limits, but if you are a researcher and do not naturally describe your work in terms of the open problems you are trying to close, you might wonder whether you are tough enough on yourself.



Mark Guzdial
CS Classes Have Different Results than Laboratory Experiments—
Not in a Good Way

<http://bit.ly/1UUrOUu>
March 29, 2016

I have collaborated with Lauren Margulieux on a series of experiments and

papers around using subgoal labeling to improve programming education. She has just successfully defended her dissertation. I describe her dissertation work, and summarize some of her earlier findings, in the blog post at <http://bit.ly/23bxRWd>.

She had a paragraph in her dissertation’s methods section that I just flew by when I first read it:

Demographic information was collected for participants’ age, gender, academic field of study, high school GPA, college GPA, year in school, computer science experience, comfort with computers, and expected difficulty of learning App Inventor because they are possible predictors of performance (Rountree, Rountree, Robins, & Hannah, 2004; see Table 1). These demographic characteristics were not found to correlate with problem solving performance (see Table 1).

Then I realized her *lack* of result was a pretty significant result.

I asked her about it at the defense. She collected all these potential predictors of programming performance in all the experiments. Were they *ever* a predictor of the experiment outcome? She said she once, out of eight experiments, found a weak correlation between high school GPA and performance. In all other cases, “these demographic characteristics were not found to correlate with problem solving performance” (to quote her dissertation).

There has been a lot of research into what predicts success in programming classes. One of the more controversial claims is that a mathematics background is a prerequisite for learning programming. Nathan Ensmenger suggests the studies show a correlation between mathematics background and success in programming *classes*, but not in programming *performance*. He suggests overemphasizing mathematics has been a factor in the decline in diversity in computing (see <http://bit.ly/1ql27jD> about this point).

These predictors are particularly important today. With our burgeoning undergraduate enrollments, programs are looking to cap enrollment using factors like GPA to decide who gets to stay in CS (see Eric Roberts’ history of enrollment caps in CS at <http://bit.ly/2368RmV>). Margulieux’s results suggest choosing who gets into CS based on GPA might

be a bad idea. GPA may not be an important predictor of success.

I asked Margulieux how she might explain the difference between her experimental results and the classroom-based results. One possibility is that there *are* effects of these demographic variables, but they are too small to be seen in short-term experimental settings. A class experience is the sum of many experiment-size learning situations.

There is another possibility Margulieux agrees could explain the difference between classrooms and laboratory experiments: we may teach better in experimental settings than we do in classes. Lauren has almost no one dropping out of her experiments, and she has measurable learning. *Everybody* learns in her experiments, but some learn more than others. The differences cannot be explained by any of these demographic variables.

Maybe characteristics like “participants’ age, gender, academic field of study, high school GPA, college GPA, year in school, computer science experience, comfort with computers, and expected difficulty of learning” programming are predictors of success in programming *classes* because of *how* we teach programming *classes*. Maybe if we taught differently, more of these students would succeed. The predictor variables may say more about our teaching of programming than about the challenge of learning programming.

Reader’s comment:

Back in the 1970s when I was looking for my first software development job, companies were using all sorts of tests and “metrics” to determine who would be a good programmer. I’m not sure any of them had any validity. I don’t know that we have any better predictors today. In my classes these days, I see lots of lower-GPA students who do very well in computer science classes. Maybe it is how I teach. Maybe it is something else (interest?), but all I really know is that I want to learn better how to teach.

—Alfred Thompson

John Langford is a Principal Researcher at Microsoft Research New York. **Bertrand Meyer** is a professor at ETH Zurich. **Mark Guzdial** is a professor at the Georgia Institute of Technology.

The Key to Privacy

40 years ago, Whitfield Diffie and Martin E. Hellman introduced the public key cryptography used to secure today's online transactions.

IT WAS UNUSUAL for Martin Hellman, a professor of electrical engineering at Stanford University, to present two papers on cryptography at the International Symposium on Information Theory in October 1977. Under normal circumstances, Steve Pohlig or Ralph Merkle, the doctoral students who also had worked on the papers, would have given the talks, but on the advice of Stanford's general counsel, it was Hellman who spoke.

The reason for the caution was that an employee of the U.S. National Security Agency, J.A. Meyer, had claimed publicly discussing their new approach to encryption would violate U.S. law prohibiting the export of weapons to other countries. Stanford's lawyer did not agree with that interpretation of the law, but told Hellman it would be easier for him to defend a Stanford employee than it would be to defend graduate students, so he recommended Hellman give the talk instead.

Whitfield Diffie, another student of Hellman's who says he was a hippie with "much more anti-societal views then," had not been scheduled to present a paper at the conference, but came up with one specifically to thumb his nose at the government's claims. "This was just absolute non-

sense, that you could have laws that could affect free speech," Diffie says. "It was very important to defy them."

In the end, no one was charged with breaking any laws, though as Hellman, now professor emeritus, recalls, "there was a time there when it was pretty dicey." Instead, the researchers' work started to move the field of cryptography into academia and the commercial world, where the cutting edge had belonged almost exclusively to government researchers doing classified work.

Diffie and Hellman wrote a paper in 1976, "New Directions in Cryptography," introducing public key cryptography that still prevails in secure online transactions today. As a result, they

The researchers' work started to move the field of cryptography into the realm of academia and the commercial world.

have been named the 2015 recipients of the ACM A.M. Turing Award.

Public key cryptography arose as the solution to two problems, says Diffie, former vice president and chief security officer at Sun Microsystems. One was the problem of sharing cryptographic keys. It was possible to encrypt a message, but for the recipient to decrypt it, he would need the secret key with which it was encrypted. The sender could physically deliver the secret key by courier or registered mail, but with millions of messages, that quickly becomes unwieldy. Another possibility would be to have a central repository of keys and distribute them as needed. That is still difficult, and not entirely satisfactory, Diffie says. "I was so countercultural that I didn't regard a call as secure if some third party knew the key."

Meanwhile, Diffie's former boss, John McCarthy, a pioneer in the field of artificial intelligence, had written about future computer systems in which people could use home terminals to buy and sell things; that would require digital signatures that could not be copied, in order to authenticate the transactions.

Both problems were solved with the idea of a public key. It is possible to generate a pair of complementary cryptographic keys. A person who wants to receive a message generates the pair



and makes one public; then a sender can use that public key to encrypt a message, but only the person with the private key, which does not have to be sent anywhere, can decrypt it.

Hellman compares it to a box with two combination locks, one to lock the box and the other to unlock it. Alice, the sender, and Bob, the receiver, each generate a pair of keys and make one public. Alice puts a message in the “box,” then locks it with her secret key, guaranteeing its authenticity since only she knows how to do that. She then places that locked box inside a larger one, which she locks with Bob’s public key. When Bob gets the box, he uses his private key to get past the outer box, and Alice’s public key to open the inner box and see the message.

Hellman and Diffie, building on an approach developed by Merkle, later came up with a variation on the scheme now called the Diffie-Hellman Key Exchange (though Hellman argues Merkle’s name should be on it as well). In this version, the box has a hasp big enough for two locks. Alice places her message in the box and locks it with a lock to which only she knows the combination, then

“Cryptography has really blossomed since the publication of their paper. It’s become the key tool of the information age.”

sends it to Bob. Bob cannot open it, nor can anyone who intercepts it en route, but he adds his own lock and sends it back. Alice then takes off her lock and sends the box back to Bob with only his lock securing it. On arrival, he can open it. In the Internet world, that translates to a commutative one-way function that allows Alice and Bob to create a common key in a fraction of a second. While an eavesdropper, in theory, could compute the same key from what he hears, that would take millions of years.

Ron Rivest, an Institute Professor in the Massachusetts Institute of Technology’s Computer Science and Artificial Intelligence Laboratory, calls the duo’s impact on the field revolutionary. “Cryptography has really blossomed since the publication of their paper,” he says. “It’s become a key tool of the information age.” Rivest, with his colleagues Adi Shamir and Leonard Adleman, developed the first practical implementation of public key encryption, stimulated, Rivest says, by Diffie and Hellman’s paper. Rivest, Shamir, and Adleman were awarded the ACM A.M. Turing Award for that work in 2002.

The Turing Award carries a \$1 million prize, which Diffie and Hellman will split. Diffie says he plans to use his half of the award to pursue research on the history of cryptography. Hellman and his wife, Dorothea, will use the money, and the attendant publicity, to bring attention to their forthcoming book about how they transformed an almost-failed marriage into one in which they have reclaimed the love they felt when they first met, and how that same approach can be used to rescue the world from the risk posed by nuclear weapons.

If young people want to go into the field of cryptography, there are three great problems for them to tackle, Diffie says: cryptography resistant to quantum computing; proof of the computational complexity of cryptosystems; and homomorphic encryption that would allow computations to be carried out on encrypted data.

Hellman encourages people to take risks and not wait to know everything they think they should know before launching a project. “When I first started working in cryptography, my colleagues all told me I was crazy,” he says. “My advice, is don’t worry about doing something foolish.” **C**

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

© 2016 ACM 0001-0782/16/06 \$15.00



Watch the Turing recipients discuss their work in this exclusive *Communications* video. <http://cacm.acm.org/videos/the-key-to-privacy>



Martin E. Hellman (left) and Whitfield Diffie.

What Happens When Big Data Blunders?

Big data is touted as a cure-all for challenges in business, government, and healthcare, but as disease outbreak predictions show, big data often fails.

YOU CANNOT BROWSE technology news or dive into an industry report without typically seeing a reference to “big data,” a term used to describe the massive amounts of information companies, government organizations, and academic institutions can use to do, well, anything. The problem is, the term “big data” is so amorphous that it hardly has a tangible definition.

While it is not clearly defined, we can define it for our purposes as: the use of large datasets to improve how companies and organizations work.

While often heralded as The Next Big Thing That Will Cure All Ills, big data can, and often does, lead to big blunders. Nowhere is that more evident than its use in forecasting outbreaks and spread of diseases.

An influenza forecasting service pioneered by Google employed big data—and failed spectacularly to predict the 2013 flu outbreak. Data used to prognosticate Ebola’s spread in 2014 and early 2015 yielded wildly inaccurate results. Similarly, efforts to predict the spread of avian flu have run into problems with data sources and interpretations of those sources.

These initiatives failed due to a combination of big data inconsistencies and human errors in interpreting that data. Together, those factors lay bare how big data might not be the solution to every problem—at least, not on its own.

Big Data Gets the Flu

Google Flu Trends was an initiative the Internet search giant began in 2008. The program aimed to better predict flu outbreaks using Google search data and information from the U.S. Centers for Disease Control and Prevention (CDC).

The big data from online searches, combined with the CDC’s cache of dis-



ease-specific information, represented a huge opportunity. Many people will search online the moment they feel a bug coming on; they look for information on symptoms, stages, and remedies. Combined with the CDC’s insights into how diseases spread, the knowledge of the numbers and locations of people seeking such information could theoretically help Google predict where and how severely the flu would strike next—before even the CDC could. In fact, Google theorized it could beat CDC predictions by up to two weeks.

The success of Google Flu Trends would have big implications. In the last three decades, thousands have died from influenza-related causes, says the CDC, while survivors can face severe health issues because of the disease. Also, many laid up by the flu consume the time, energy, and resources of healthcare organizations. Any improvement in forecasting outbreaks could save lives and dollars.

However, over the years, Google Flu Trends consistently failed to predict flu cases more accurately than the CDC. After the program failed to predict the 2013 flu outbreak, Google quietly shuttered the program.

David Lazer and Ryan Kennedy studied why the program failed, and found key lessons about avoiding big data blunders.

The Hubris of Humans

Google Flu Trends failed for two reasons, say Lazer and Kennedy: big data hubris, and algorithmic dynamics.

Big data hubris means Google researchers placed too much faith in big data, rather than partnering big data with traditional data collection and analysis. Google Flu Trends was built to map not only influenza-related trends, but also seasonal ones. Early on, engineers found themselves weeding out false hits concerned with seasonal, but not influenza-related, terms—such as those related to high school basketball season. This, say Lazer and Kennedy, should have raised red flags about the data’s reliability. Instead, it was thought the terms could simply be removed until the results looked sound.

As Lazer and Kennedy say in their article in *Science*: “Elsewhere, we have asserted that there are enormous scientific possibilities in big data. However, quantity of data does not mean that one can ignore foundational issues of measurement and construct validity and reliability and dependencies among data.”

In addition, Google itself turned out to be a major problem.

The second failure condition was one of algorithmic dynamics, or the idea that Google Flu Trends predictions were based on a commercial search algorithm that frequently changes based on Google’s business goals.

Google’s search algorithms change often; in fact, say Lazer and Kennedy, in June and July 2012 alone, Google’s algorithms changed 86 times as the firm tweaked how it returned search results in line with its business and growth goals. This sort of dynamism was not accounted for in Google Flu Trends models.

“Google’s core business is improving search and driving ad revenue,” Kennedy told *Communications*. “To do this, it is

continuously altering the features it offers. Features like recommended searches and specialized health searches to diagnose illnesses will change search prominence, and therefore Google Flu Trends results, in ways we cannot currently anticipate." This uncertainty skewed data in ways even Google engineers did not understand, even skewing the accuracy of predictions.

Google is not alone: assumptions are dangerous in other types of outbreak prediction. Just ask the organizations that tried to predict Ebola outbreaks in 2014.

Failing to Foresee Ebola

Headlines across the globe screamed worst-case scenarios for the Ebola outbreak of 2014. There were a few reasons for that: it was the worst such outbreak the world had ever seen, and there were fears the disease could become airborne, dramatically increasing its spread. In addition, there were big data blunders.

At the height of the frenzy, according to *The Economist* (<http://economist/110HYKO>), the United Nations' public health arm, the World Health Organization (WHO), predicted 20,000 cases of Ebola—nearly 54% more than the 13,000 cases reported. The CDC predicted a worst-case scenario of a whopping 1.4 million cases. In the early days of the outbreak, WHO publicized a 90% death rate from the disease; the reality at that initial stage was closer to 70%.

Why were the numbers so wrong? There were several reasons, says Aaron King, a professor of ecology at the University of Michigan. First was the failure to account for intervention; like Google's researchers, Ebola prognosticators failed to account for changing conditions on the ground. Google's model was based on an unchanging algorithm; Ebola researchers used a model based on initial outbreak conditions. This was problematic in both cases: Google could not anticipate how its algorithm skewed results; Ebola fighters failed to account for safer burial techniques and international interventions that dramatically curbed outbreak and death-rate numbers.

"Perhaps the biggest lesson we learned is that there is far less information in the data typically available in the early stages of an outbreak than

"In the future, I hope we as a community get better at distinguishing information from assumptions," King says.

is needed to parameterize the models that we would like to be able to fit," King told *Communications*.

That was not the only mistake made, says King. He argues stochastic models that better account for randomness are more appropriate for predictions of this kind. Ebola fighters used deterministic models that did not account for the important random elements in disease transmission.

"In the future, I hope we as a community get better at distinguishing information from assumptions," King says.

Can We Ever Predict Outbreaks Accurately?

It is an open question whether models can be substantially improved to predict disease outbreaks more accurately.

Other companies want to better predict flu outbreaks after the failure of Google Flu Trends—specifically avian flu—using social media and search platforms. Companies such as Sickweather and Epidemico Inc. use algorithms and human curation to assess both social media and news outlets for flu-related information.

These efforts, however, run the same risks as previous flu and Ebola prediction efforts. Social media platforms change, and those changes do not always benefit disease researchers. In fact, says King, data collection may hold the key to better predictions.

"I suspect that our ability to respond effectively to future outbreaks will depend more on improved data collection techniques than on improvement in modeling technologies," he says.

Yet even improvements in data collection might not be enough. In addition to internal changes that affect

how data is collected, researchers must adapt their assessments of data to conditions on the ground. Sometimes, as in the case of avian flu, not even experts understand what to look for right away.

"The biggest challenge of the spring 2015 outbreak [of avian flu] in the United States was that poultry producers were initially confused about the actual transmission mechanism of the disease," says Todd Kuethe, an agricultural economist who writes on avian flu topics. "Producers initially believed it was entirely spread by wild birds, but later analysis by the USDA (U.S. Department of Agriculture) suggested that farm-to-farm transmission was also a significant factor."

No matter the type of data collection or the models used to analyze it, sometimes disease conditions change too quickly for humans or algorithms to keep up. That might doom big data-based disease prediction from the beginning.

"The ever-changing situation on the ground during emerging outbreaks makes prediction failures inevitable, even with the best models," concludes Matthieu Domenech De Celles, a postdoctoral fellow at the University of Michigan who has worked on Ebola prediction research. **C**

Further Reading

Lazer, D., and Kennedy, R. (2014) The Parable of Google Flu: Traps in Big Data Analysis. *Science*. <http://scholar.harvard.edu/files/gking/files/0314policyforumff.pdf>

Miller, K. (2014) Disease Outbreak Warnings Via Social Media Sought By U.S. *Bloomberg*. <http://www.bloomberg.com/news/articles/2014-04-11/disease-outbreak-warnings-via-social-media-sought-by-u-s>

Erickson, J. (2015) Faulty Modeling Studies Led To Overstated Predictions of Ebola Outbreak. *Michigan News*. <http://ns.umich.edu/new/releases/22783-faulty-modeling-studies-led-to-overstated-predictions-of-ebola-outbreak>

Predictions With A Purpose. *The Economist*. <http://www.economist.com/news/international/21642242-why-projections-ebola-west-africa-turned-out-wrong-predictions-purpose>

Logan Kugler is a freelance technology writer based in Tampa, FL. He has written for over 60 major publications.

© 2016 ACM 0001-0782/16/06 \$15.00

Reimagining Search

Search engine developers are moving beyond the problem of document analysis, toward the elusive goal of figuring out what people really want.

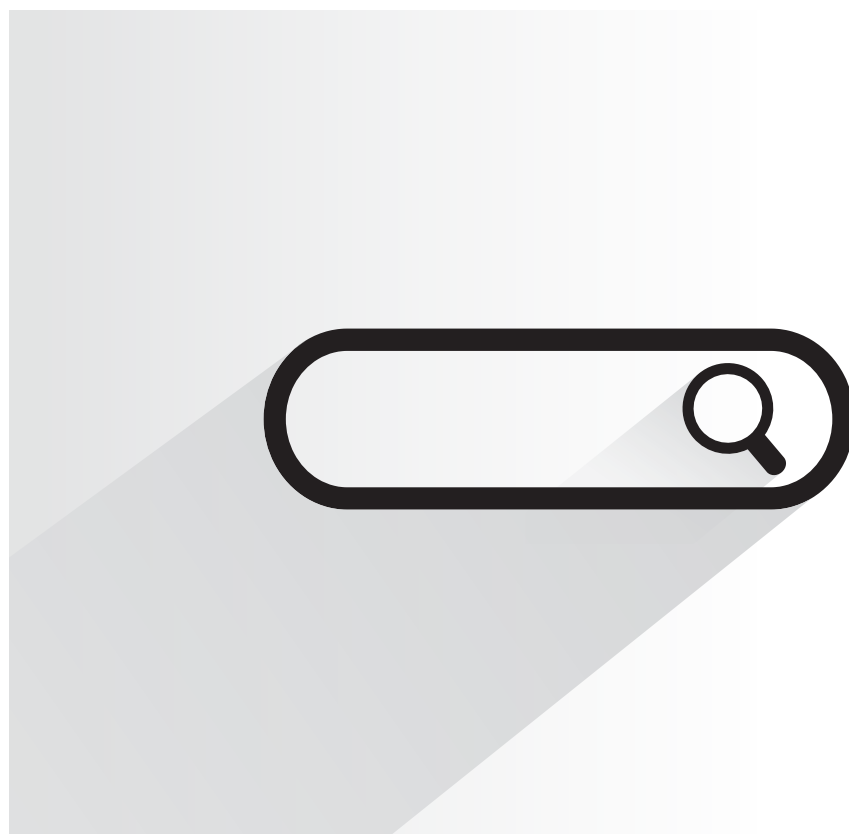
EVER SINCE GERARD SALTON of Cornell University developed the first computerized search engine (Salton's Magical Automatic Retriever of Text, or SMART) in the 1960s, search developers have spent decades essentially refining Salton's idea: take a query string, match it against a collection of documents, then calculate a set of relevant results and display them in a list. All of today's major Internet search engines—including Google, Amazon, and Bing—continue to follow Salton's basic blueprint.

Yet as the Web has evolved from a loose-knit collection of academic papers to an ever-expanding digital universe of apps, catalogs, videos, and cat GIFs, users' expectations of search results have shifted. Today, many of us have less interest in sifting through a collection of documents than in getting something *done*: booking a flight, finding a job, buying a house, making an investment, or any number of other highly focused tasks.

Meanwhile, the Web continues to expand at a dizzying pace. Last year, Google indexed roughly 60 trillion pages—up from a mere one trillion in 2008.

"As the Web got larger, it got harder to find the page you wanted," says Ben Gomes, a Google Fellow and vice president of the search giant's Core Search team, who has been working on search at Google for more than 15 years.

Today's Web may bear little resemblance to its early incarnation as a academic document-sharing tool, yet the basic format of search results has remained remarkably static over the years. That is starting to change, however, as search developers shift focus from document analysis to the even thornier challenge of trying to understand the kaleidoscope of human wants and needs that underlie billions of daily Web searches.



While document-centric search algorithms have largely focused on solving the problems of semantic analysis—identifying synonyms, spotting spelling errors, and adjusting for other linguistic vagaries—many developers are now shifting focus to the other side of the search transaction: the query itself.

By mining the vast trove of query terms that flow through Web search engines, developers are exploring new ways to model the context of inbound query strings, in hopes of improving the precision and relevance of search results.

"Before you look at the documents, you try to determine the intent," says Daniel Tunkelang, a software engineer who formerly led the search team at LinkedIn.

There, Tunkelang developed a sophisticated model for query understanding that involved segmenting incoming queries into groups by tagging relevant entities in each query, categorizing certain sequences of tags to identify the user's likely intent, and using synonym matching to further refine the range of likely intentions.

At LinkedIn, a search for "Obama" returns a link to the president's profile page, while a search for "president" returns a list of navigational shortcuts to various jobs, people, and groups containing that term. When the user selects one of those shortcuts, LinkedIn picks up a useful signal about that user's intent, which it can then use to return a highly targeted result set.

In a similar vein, a search for "Hemingway" on Amazon will return a

familiar-looking list of book titles, but a search for a broader term like “outdoors” will yield a more navigational page with links to assorted Amazon product categories. By categorizing the query—distinguishing a “known item” search from a more exploratory keyword search—Amazon tries to adapt its results based on a best guess at the user’s goal.

The widespread proliferation of structured data, coupled with advances in natural language processing and the rise of voice recognition-equipped mobile devices, has given developers a powerful set of signals for modeling intent, enabling them to deliver result formats that are highly customized around particular use cases, and to invite users into more conversational dialogues that can help fine-tune search results over time.

Web users can see a glimpse of where consumer search may be headed in the form of Google’s increasingly ubiquitous “snippets,” those highly visible modules that often appear at the top of results pages for queries on topics like sports scores, stock quotes, or song lyrics. Unlike previous incarnations of Google search results, snippets are trying to do more than just display a list of links; they are trying to answer the user’s question.

These kinds of domain-specific searches benefit from a kind of a priori knowledge of user intent. Netflix, for example, can reasonably infer most queries have something to do with mov-

By analyzing the interplay of query syntax and synonyms, Google looks for linguistic patterns that can help refine the search result.

ies or TV. Yet a general-purpose search engine like Google must work harder to gauge the intent of a few characters’ worth of text pointed at the entire Web.

Developers are now beginning to make strides in modeling the context of general Web searches, thanks to a number of converging technological trends: advances in natural language processing; the spread of location-aware, voice recognition-equipped mobile devices, and the rise of structured data that allows search engines to extract specific data elements that might once have remained locked inside a static Web page.

Consumer search engines also try to derive user intent by applying natural language processing techniques to inbound search terms.

For example, when a user enters the phrase “change a lightbulb,” the word “change” means “replace;” but if a user enters “change a monitor,” the term “change” means “adjust.”

By analyzing the interplay of query syntax and synonyms, Google looks for linguistic patterns that can help refine the search result. “We try to match the query language with the document language,” says Gomes. “The corpus of queries and the corpus of documents come together to give us a deeper understanding of the user’s intent.”

Beyond the challenges of data-driven query modeling, some search engine developers are finding inspiration by looking beyond their search logs and turning their gaze outward to deepen their understanding of real-life users “in the wild.”

“Qualitative research is great to generate insight and hypotheses,” says Tunkelang, who sees enormous potential in applying user experience (UX) research techniques to assess the extent to which users may trust a particular set of search results, or exploring why they may not choose to click on a particular link in the results list. Qualitative research can also shed light on deeper emotional needs that may be difficult to ascertain through data analysis alone.

At Google, the search team runs an ongoing project called the Daily Information Needs study, in which 1,000 volunteers in a particular region receive a ping on their smartphones up

Milestones

Computer Science Awards, Appointments

PAPADIMITRIOU AWARDED VON NEUMANN MEDAL

IEEE has honored Christos H. Papadimitriou, C. Lester Hogan Professor in the Department of Electrical Engineering and Computer Science at the University of California, Berkeley, with the 2016 John von Neumann Medal “for providing a deeper understanding of computational complexity and its implications for approximation algorithms, artificial intelligence, economics, database theory, and biology.”

Papadimitriou, who has taught at Harvard, the Massachusetts Institute of

Technology, the National Technical University of Athens, Stanford University, and the University of California at San Diego, is the author of the textbook *Computational Complexity*, which is widely used in the field of computational complexity theory. He also co-authored the textbook *Algorithms* with Sanjoy Dasgupta and Umesh Vazirani, and the graphic novel *Logicomix* with Apostolos Doxiadis.

The IEEE John von Neumann Medal is awarded for outstanding achievements in computer-related science and technology.

ACM CITES PERROT FOR VISION, LEADERSHIP

ACM has named Ron Perrot of the Queen’s University Belfast/Oxford e-Research Centre recipient of the 2015 ACM Distinguished Service Award “for providing vision and leadership in high-performance computing and e-science, championing new initiatives and advocating collaboration among interested groups at both national and international levels.”

Perrott was cited for providing vision and leadership in high-performance computing and e-science, championing new

initiatives, and advocating collaboration among interested groups at the national and international levels. He has been an effective advocate for high-performance and grid computing in Europe since the 1970s, working tirelessly and successfully with academic, governmental, and industrial groups to convince them of the importance of developing shared resources for high-performance computing at both national and regional levels.

Perrot is a Fellow of ACM, IEEE, and the British Computing Society.

to eight times per day to report on what kind of information they are looking for that day—not just on Google, but anywhere. Insights from this study have helped Google seed the ideas for new products such as Google Now.

Researchers at Microsoft recently conducted an ethnographic study that pointed toward five discrete modes of Web search behavior:

- **Respite:** taking a break in the day's routine with brief, frequent visits to a familiar set of Web sites;
- **Orienting:** frequent monitoring of heavily-used sites like email providers and financial services;
- **Opportunistic use:** leisurely visits to less-frequented sites for topics like recipes, odd jobs, and hobbies;
- **Purposeful use:** non-routine usage scenarios, usually involving time-limited problems like selling a piece of furniture, or finding a babysitter, and
- **Lean-back:** consuming passive entertainment like music or videos.

Each of these modes, the authors argue, calls for a distinct mode of on-screen interaction, “to support the construction of meaningful journeys that offer a sense of completion.”

As companies begin to move away from the one-size-fits-all model of list-style search results, they also are becoming more protective of the underlying insights that shape their presentation of search results.

“One irony is that as marketers have gotten more sophisticated, the amount of data that Google is sharing with its marketing partners has actually diminished,” says Andrew Frank, vice president of research at Gartner. “It used to be that if someone clicked on an organic link, you could see the search terms they used, but over the past couple of years, Google has started to suppress that data.”

Frank also points to Facebook as an example of a company that has turned query data into a marketing asset, by giving marketers the ability to optimize against certain actions without having to target against particular demographics or behaviors.

As search providers continue to try to differentiate themselves based on a deepening understanding of query intent, they will also likely focus on capturing more and more information about the context surrounding a partic-

ular search, such as location, language, and the history of recent search queries. Taken together, these cues will provide sufficient fodder for increasingly predictive search algorithms.

Tunkelang feels the most interesting unsolved technical problem in search involves so-called query performance prediction. “Search engines make dumb mistakes and seem blissfully unaware when they are doing so,” says Tunkelang.

“In contrast, we humans may not always be clever, but we’re much better at calibrating our confidence when it comes to communication. Search engines need to get better at query performance prediction—and better at providing user experiences that adapt to it.”

Looking even further ahead, Gomes envisions a day when search engines will get so sophisticated at modeling user intent that they will learn to anticipate users’ needs well ahead of time. For example, if the system detects you have a history of searching for Boston Red Sox scores, your mobile phone could greet you in the morning with last night’s box score.

Gomes thinks this line of inquiry may one day bring search engines to the cusp of technological clairvoyance. “How do we get the information to you before you’ve even asked a question?” ■

Further Reading

Bailey, P., White, R.W., Liu, H., and Kumaran, G., Mining Historic Query Trails to Label Long and Rare Search Engine Queries. *ACM Transactions on the Web*. Volume 4 Issue 4, Article 15 (September 2010), <http://dx.doi.org/10.1145/1841909.1841912>

Lindley, S., Meek, S., Sellen, A., and Harper, R., ‘It’s Simply Integral to What I do.’ Enquiries into how the Web is Weaved into Everyday Life, WWW 2012, <http://research.microsoft.com/en-us/people/asellen/wwwmodes.pdf>

Salton, G., *The SMART Retrieval System—Experiments in Automatic Document Processing*, Prentice-Hall, Inc., Upper Saddle River, NJ, 2012

Vakkari, P., Exploratory Searching as Conceptual Exploration, Microsoft Research, <http://bit.ly/1N3rI3x>

Alex Wright is a writer and information architect based in Brooklyn, NY.

© 2016 ACM 0001-0782/16/06 \$15.00

ACM Member News

A “LITTLE DIFFERENT” CAREER TRAJECTORY



“It’s a little different,” says Julia Hirschberg, Percy K. and Vida L.W. Hudson

Professor of Computer Science and Chair of the Computer Science Department at Columbia University, of her career trajectory.

Hirschberg majored in history as an undergraduate, earning a Ph.D. in 16th century Mexican social history at the University of Michigan at Ann Arbor. While teaching history at Smith College, she discovered artificial intelligence techniques were useful in building social networks of 16th century colonists from “fuzzy” data. She soon decided computer science was even more exciting than history and went back to school, earning a doctorate in computer science from the University of Pennsylvania in 1985.

“None of my career decisions have been carefully planned. You often see opportunities you never dreamed would be possible.”

As a result of her thesis work, Hirschberg met researchers at Bell Laboratories. She went to work there in 1985, first working in test-to-speech synthesis, then launching the Human-Computer Interface Research Department in 1994, and moving with Bell to AT&T Laboratories.

Hirschberg started teaching at Columbia in 2002, and became chair of the Computer Science Department in 2012. Her major research area is computational linguistics; her current interests include deceptive speech and spoken dialogue systems.

“One of the things I think of when I tell young women about my career is that many opportunities arise,” Hirschberg says. “I never knew as an undergraduate that I would become a computer scientist, let alone chairing a computer science department at Columbia. You make some decisions, but they are not necessarily decisions for life.”

—John Delaney

What's Next for Digital Humanities?

New computational tools spur advances in an evolving field.

IN 1946, AN Italian Jesuit priest named Father Roberto Busa conceived of a project to index the works of St. Thomas Aquinas word by word. There were an estimated 10 million words, so the priest wondered if a computing machine might help. Three years later, he traveled to the U.S. to find an answer, eventually securing a meeting with IBM founder Thomas J. Watson. Beforehand, Busa learned Watson's engineers had already informed him the task would be impossible, so on his way into Watson's office, he grabbed a small poster from the wall that read, "The difficult we do right away; the impossible takes a little longer." The priest showed the executive his own company's slogan, and Watson promised IBM's cooperation.

"The impossible" took roughly three decades, but that initial quest also marked the beginning of the field now known as Digital Humanities. Today, digital humanists are applying advanced computational tools to a wide range of disciplines, including literature, history, and urban studies. They are learning programming languages, generating dynamic three-dimensional (3D) re-creations of historic city spaces, developing new academic publishing platforms, and producing scholarship.

The breadth of the field has led to something of an identity crisis. In fact, there is an annual Day of Digital Humanities (which was April 8 this year), during which scholars publish details online about the work they are conducting on that particular date. The goal is to answer the question, "Just what do digital humanists really do?"

As it turns out, there are many different answers.

Distant Reading

Digital Humanities is most frequently associated with the computational



Father Roberto Busa, whose project to index the works of St. Thomas Aquinas marked the beginning of Digital Humanities.

analysis of text, from the Bible to modern literature. One common application is distant reading, or the use of computers to study hundreds or thousands of books or documents rather than having a human pore over a dozen.

Consider Micki Kaufman, a Ph.D. candidate at The Graduate Center, City University of New York (CUNY), who decided to study the digitized correspondence of Henry Kissinger. This was no small task; she was faced with transcripts of more than 17,500 telephone calls and 2,200 meetings. Adding to the challenge was the fact that some of the materials had been redacted for national security reasons. She realized by taking a computational approach, she could glean insights both into the body of documents as a whole and the missing material.

In one instance, Kaufman used a machine-reading technique combining word collocation and frequency analysis to scan the texts for the words "Cambodia" and "bombing," and to track how far apart they appear within the text. A statement such as "We are

bombing Cambodia" would have a distance of zero, whereas the result might be 1,000 if the terms are separated by several pages. Kaufman noticed the words tended to be clustered together more often in telephone conversations, suggesting Kissinger believed he had greater privacy on the phone, relative to the meetings, and therefore spoke more freely. Furthermore, the analysis offered clues to what had been redacted, as it turned up major gaps in the archive—periods during which the terms did not appear together—when the bombing campaign was known to be active.

Overall, Kaufman was able to study the archive through a different lens, and found patterns she might not have detected through a laborious reading of each file. "You get the long view," says Kaufman. "You can ask yourself about behavioral changes and positional changes in ways that would have required the reading of the entire set."

The computer-aided approach of distant reading has also started to move beyond texts. One example is the work of the cultural historian Lev Manovich, also of The Graduate Center, CUNY, who recently subjected a dataset of 6,000 paintings by French Impressionists to software that extracted common features in the images and grouped them together. Manovich and his colleagues found more than half of the paintings were reminiscent of the standard art of the day; Impressionist-style productions, on the other hand, represented only a sliver of the total works.

A New Way of Seeing

That sort of finding would be of interest to any Impressionist historian, not just those with a digital bent, and according to University of Georgia historian Scott Nesbit, this is a critical distinction. Digital humanists have

their own dedicated journals and conferences, but to Nesbit this might not be the best approach going forward. “I don’t see Digital Humanities as its own discipline,” he says. “We’re humanists who use certain methods and certain tools to try to understand what’s going on in our discipline and ask questions in ways we hadn’t been asking before.”

When Nesbit set out to analyze the post-emancipation period during the U.S. Civil War, he wanted to look at exactly how enslaved people became free, and specifically how the movement of the anti-slavery North’s Union Army impacted that process. “We wanted to come up with a way to see what emancipation actually looked like on the ground,” he says.

Nesbit and his colleagues extracted data from both U.S. Census results and advertisements of slave owners looking for their freed servants. They built a Geographic Information System (GIS) map of the region, and then overlaid the apparent tracks of the freed slaves with the movements of the Union Army at the time. What they found surprised them: there were the expected spikes in the number of freed slaves escaping when the army arrived, but these advances apparently did not inspire everyone to seek freedom. The people fleeing north were predominantly men; of the few advertisements seeking runaway women that do appear during these periods, the data suggests they escaped to the city instead. “There are a number of possible reasons for this,” Nesbit says, “one of them being that running toward a group of armed white men might not have seemed like the best strategy for an enslaved woman.”

This gender-based difference to the workings of emancipation was a new insight relevant to any historian of the period—not just the subset who prefer digital tools. While Nesbit might have spotted the same trend through exhaustive research, the digital tools made it much easier to see patterns in the data. “It was important to visualize these in part so I could see the spatial relationships between armies and the actions of enslaved people,” Nesbit says.

The art historians, architects, and urban studies experts behind a project called Visualizing Venice hope

for similarly surprising results. This collaboration between academics at Duke University, the University of Venice, and the University of Padua generates 3D representations of specific areas within the famed city, and how the buildings, public spaces, and even interior designs of its structures have changed over the centuries. The researchers create accurate digital representations of various buildings in their present form, using laser radar scanning and other tools, then draw upon historical paintings, architectural plans, civic documents, and more to effectively roll back the clock and trace each structure’s evolution over time. The animations allow researchers to watch buildings grow and change in response to the evolving city, but they are not just movies; they are annotated in such a way that it is possible to click through a feature to see the historical document(s) on which it is based.

Beyond the Computationally Inflected

While the goal of Visualizing Venice is in part to produce scholarship, other experts argue Digital Humanities also encompass the development of tools designed to simplify research.

The programmer and amateur art historian John Resig, for example, found himself frustrated at the difficulty of searching for images of his favorite style of art, Japanese woodblock prints. He wrote software that scours the digital archives of a museum or university and copies relevant images and their associated metadata to his site. Then he applied the publicly available Match-Engine software tool, which scans these digital reproductions for similarities and finds all the copies of the same print, so he could organize his collection by image. In short, he developed a simple digital way for people to find the physical locations of specific prints.

At first, Resig says, academics did not take to the tool. “There was one scholar who said, ‘That sounds useful, but not for me, because I’m already an expert,’” Resig recalls. “A year later, this scholar came to me and said, ‘I’m so glad you built this website. It saves me so much time!’”

This type of contribution has become commonplace in the field of Archaeology. For example, the Codifi

software platform, developed in part by archaeologists from the University of California, Berkeley, is designed to reduce field researchers’ dependence on paper, giving them an easier and more scalable way to collect and organize images, geospatial information, video, and more. Archaeologists also have proven quick to explore the potential of even more advanced technologies, from 3D printers that generate reproductions of scanned artifacts to the possibility of using low-cost drones equipped with various sensors as a new way of analyzing dig sites.

Yet archaeologists who engage in this kind of work are rarely considered digital humanists, or even digital archaeologists. Archaeology was so quick to adopt computational tools and methods and integrate them into the practice of the discipline that the digital aspect has integrated with the field as a whole. This might be a kind of roadmap for digital humanists in other disciplines to follow.

Matthew Gold, a digital humanist at The Graduate Center, CUNY, suggests the time is right for such a shift. “What we’re seeing now is a maturation of some of the methods, along with an effort by digital humanists to test their claims against the prevailing logic in their field, so that it’s not just computationally inflected work off to the side,” Gold says. “The field is at an interesting moment.” ■

Further Reading

Gold, M. (Ed.)
Debates in the Digital Humanities,
The University of Minnesota Press, 2016.

Berry, D.M. (Ed.)
Understanding Digital Humanities,
Palgrave Macmillan, 2012.

Nesbit, S.
Visualizing Emancipation: Mapping the End of Slavery in the American Civil War, in Computation for Humanity: Information Technology to Advance Society (New York: Taylor & Francis), 427-435.

Moretti, F.
Graphs, Maps, Trees,
New Left Review, 2003.

Visualizing Venice Video:
<http://bit.ly/24f5bgJ>

Gregory Mone is a Boston, MA-based science writer and children’s novelist.

© 2016 ACM 0001-0782/16/06 \$15.00

Inside Risks

The Risks of Self-Auditing Systems

Unforeseen problems can result from the absence of impartial independent evaluations.

OVER TWO DECADES ago, NIST Computer Systems Laboratory's Barbara Guttman and Edward Roback warned that "the essential difference between a self-audit and an external audit is objectivity."⁶ In that writing, they were referring to internal reviews by system management staff, typically for purposes of risks assessment—potentially having inherent conflicts of interest, as there may be disincentives to reveal design flaws that could pose security risks. In this column, we raise attention to the *additional* risks posed by reliance on information produced by electronically self-auditing sub-components of computer-based systems. We are defining such self-auditing devices as being those that display internally generated data to an independent external observer, typically for purposes of ensuring conformity and/or compliance with particular range parameters or degrees of accuracy.

Our recent interest in this topic was sparked by the revelations regarding millions of Volkswagen vehicles whose

emission systems had been internally designed and manufactured such that lower nitrogen dioxide levels would be produced and measured during the inspection-station testing (triggered by the use of the data port) than would occur in actual driving. In our earlier writings, we had similarly warned about voting machines potentially being set to detect election-day operations, such that the pre-election testing would show results consistent with practice ballot inputs, but the actual election-day ballots would not be tabulated accurately. These and other examples are described further in this column.

Issues

We are not suggesting that all self-auditing systems are inherently bad. Our focus is on the risks of explicit reliance *only* on internal auditing, *to the exclusion of any independent external oversight*. It is particularly where self-auditing systems have end-to-end autonomous checking or only human interaction with insiders, that unbiased external observation becomes unable to influence or detect flaws with the imple-

mentation and operations with respect to the desired and expected purposes.

Although many self-auditing systems suffer from a lack of sufficient transparency and external visibility to ensure trustworthiness, the expedience and the seeming authority of results can inspire false confidence. More generally, the notion of *self-regulation* poses the risk of degenerating into *no regulation whatsoever*, which appears to be the case with respect to self-auditing.

By auditing, we mean *systematic examination and verification of accounts, transaction records (logs), and other documentation, accompanied by physical inspection (as appropriate), by an independent entity*. In contrast, self-auditing results are typically internally generated, but are usually based on external inputs by users or other devices. The self-audited aggregated results typically lack a verifiable correspondence of the outputs with the inputs. As defined, such systems have no trustworthy independent checks-and-balances. Worse yet, the systems may be proprietary or covered by trade-secret protection that



Digital Meters

The relative inaccuracy of self-calibrated (or merely factory-set) meters is often neglected in electronic measurement and design. Self-calibration can be considered to be a form of self-auditing when performed to a presumed reliable reference source. Calibration is also highly dependent on the specific applications. For example, while a 5% error rate may not be of tremendous concern when measuring a 5-volt source, at higher test levels the disparity can become problematic. There is also the error of perception that comes with digital displays, where precision may be misinterpreted as accuracy. Engineers have been shown to have a propensity toward overly trusting trailing digits in a numerical read-out, when actually analog meters can provide less-misleading relative estimates.⁸

Many concerns are raised as we become increasingly dependent on health-monitoring devices. For example, millions of diabetics test their blood glucose levels each day using computerized meters. System accuracy for such consumer-grade devices is recommended to be within 15 mg/dl as compared with laboratory results, yet experimental data shows that in the low-blood sugar range (≤ 75 mg/dl), some 5% of these personal-use meters will fail to match the (presumably more stringent) laboratory tests. Reliance on results that show higher than actual values in the low range (where percentages are most critical) may result in the user's failure to take remedial action or seek emergency medical attention, as appropriate. Many users assume the meters are accurate,

explicitly precludes external inspection and validation.

Trade secrecy is often used to maintain certain intellectual property protections—in lieu of copyright and/or patent registration. It requires proofs of strict secrecy controls, which are inherently difficult to achieve in existing systems. Trade-secrecy protection can extend indefinitely, and is often used to conceal algorithms, processes, and software. It can thwart detection of illicit activity or intentional alteration of reported results.

Relying on internally generated audits creates numerous risks across a broad range of application areas, especially where end-to-end assurance is desired. In some cases, even internal audits are lacking altogether. The risks may include erroneous and compromised results, opportunities for serious misuse, as well as confusions between precision and accuracy.

Systemic Problems

Of course, the overall problems are much broader than just those relating to inadequate or inappropriately com-

promised internal auditing and the absence of external review.

Of considerable relevance to networked systems that should be trustworthy is a recent paper² that exposes serious security vulnerabilities resulting from composing implementations of apparently correctly specified components. In particular, the authors of that paper examine the client-side and server-side state diagrams of the Transport Layer Security (TLS) specification. The authors show that approximately a half-dozen different popular TLS implementations (including OpenSSL and the Java Secure Socket Extension JSSE) introduce unexpected security vulnerabilities, which arise as emergent properties resulting from the composition of the client-side and server-side software. This case is an example of an open source concept that failed to detect some fundamental flaws—despite supposed many-eyes review. Here, we are saying the self-auditing is the open-source process itself. This research illustrates some of the risks of ad hoc composition, the underlying lack of predictability that can result, and the lack of auditing sufficient for correctness and security. However, their paper addresses only the tip of the iceberg when it comes to exploitable vulnerabilities of open source systems.

and are unaware that periodic testing should be performed using a control solution (the hefty price of which is often not covered by health insurance). In actuality, since the control-solution test uses the same meter and is not a wholly independent comparison (for example, with respect to a laboratory test), it too may not provide sufficient reliability to establish confidence of accuracy.

End-to-End System Assurance

The security literature has long demonstrated that embedded testing mechanisms in electronic systems can be circumvented or designed to provide false validations of the presumed correctness of operations. Proper end-to-end system design (such as with respect to Common Criteria and other security-related standards) is intended to ferret out such problems and provide assurances that results are being accurately reported. Unfortunately, most systems are not constructed and evaluated against such potentially stringent methodologies.

Yet, even if such methods were applied, all of the security issues may not be resolved, as was concluded in a SANS Institute 2001 white paper.¹ The author notes that the Common Criteria “can only assist the IT security communities to have the assurance they need and may push the vendor and developer for [a] better security solution. IT security is a process, which requires the effort from every individual and management in every organization. It is not just managing the risk and managing the threat; it is the security processes of Assessment, Prevention, Detection and Response; it is a cycle.” Rebecca Mercuri also points out⁷ that certain requirements cannot be satisfied simultaneously (such as, a concurrent need for system integrity and user privacy along with assuredly correct auditability), whereas the standards fail to mitigate or even address such design conflicts.

The Volkswagen Case and Its Implications

Security professionals are well aware that the paths of least resistance (such as the opportunities and knowledge provided to insiders) often form the best avenues for system exploits. These truths were underscored when Volkswagen announced in September 2015

Relying on internally generated audits creates numerous risks across a broad range of application areas.

“that it would halt sales of cars in the U.S. equipped with the kind of diesel motors that had led regulators to accuse the German company of illegally [creating] software to evade standards for reducing smog.”⁵

While Volkswagen’s recall appeared at first to be voluntary, it had actually been prompted by investigations following a March 2014 Emissions Workshop (co-sponsored by the California Air Resources Board and the U.S. Environmental Protection Agency (EPA), among others). There, a West Virginia University research team working under contract for the International Council on Clean Transportation (ICCT, a European non-profit) provided results showing the self-tested data significantly underrepresented what occurred under actual driving conditions. These revelations eventually led to a substantial devaluation of Volkswagen stock prices and the resignations of the CEO and other top company officials, followed by additional firings and layoffs. Pending class-action and fraud lawsuits and fines promise to be costly in the U.S. and abroad.

Ironically, the report⁹ was originally intended to support the adoption of the presumably strict U.S. emissions testing program by European regulators, in order to further reduce the release of nitrogen oxides into the air. Since the university researchers did not just confine themselves to automated testing, but actually drove the vehicles on-road, they were able to expose anomalous results that were as much as 40 times what is allowed by the U.S. standard defined by the Clean Air Act. The EPA subsequently recalled seven vehicle models dating from 2009–2015, including approximately 500,000 vehicles in the

U.S.; Germany ordered recall of 2.4M vehicles. Extensive hardware and software changes are required to effect the recall modifications. Still, the negative environmental impacts will not be fully abated, as the recalls are anticipated to result in poorer gas mileage for the existing Volkswagen diesel vehicles.

Election Integrity

An application area that is particularly rife with risks involves Direct Recording Electronic (DRE) voting systems—which are self-auditing. These are end-to-end automated systems, with results based supposedly entirely on users’ ballot entries. Aggregated results over multiple voters may not have assured correspondence with the inputs. Most of the commercial systems today lack independent checks and balances, and are typically proprietary and prohibited from external validation.

Reports of voters choosing one candidate and seeing their selection displayed incorrectly have been observed since the mid-1990s. This occurs on various electronic balloting systems (touchscreen or push-button). However, what happens when votes are recorded internally (or in processing optically scanned paper ballots) inherently lacks any independent validation. For example, Pennsylvania certified a system even after videotaping a vote-flipping incident during the state’s public testing. The questionable design and development processes of these systems—as well as inadequate maintenance and operational setup—are known to result in improper and unchecked screen alignment and strangely anomalous results.

Some research has been devoted to end-to-end cryptographic verification that would allow voters to demonstrate their choices were correctly recorded and accurately counted.⁴ However, this concept (as with Internet voting) enables possibilities of vote buying and selling. It also raises serious issues of the correctness of cryptographic algorithms and their implementation, including resistance to compromise of the hardware and software in which the cryptography would be embedded.

Analogous Examples

It seems immediately obvious that the ability to rig a system so it behaves cor-

rectly *only* when being tested has direct bearing on election systems. The Volkswagen situation is a bit more sophisticated because the emissions system was actually controlled differently to produce appropriate readings whenever testing was detected. Otherwise, it is rather similar to the voting scenario, where the vendors (and election officials) want people to believe the automated testing actually validates how the equipment is operating during regular operations, thus seemingly providing some assurance of correctness. While activation of the Volkswagen stealth cheat relied on a physical connection to the testing system, one might imagine a tie-in to the known locations of emission inspection stations—using the vehicle’s GPS system—which could similarly be applied to voting machines detecting their polling place.

Election integrity proponents often point to the fact that lottery tickets are printed out by the billions each year, while voting-system vendors seem to have difficulty printing out paper ballots that can be reviewed and deposited by the voter in order to establish a paper audit trail. Numerous security features on the lottery tickets are intended to enable auditing and thwart fraud, and are in principle rather sophisticated. While the location and time of lottery ticket purchases is known and recorded, this would not be possible for elections, as it violates the secrecy of the ballot. However, it should be noted that insider lottery fraud is still possible, and has been detected.

Automatic Teller Machines (ATMs) are internally self-auditing, but this is done very carefully—with extensive cross-checking for consistency to ensure each transaction is correctly processed and there are no discrepancies involving cash. There is an exhaustive audit trail. Yet, there are still risks. For example, some ATMs have been known to crash and return the screen to the operating-system command level. Even more riskful is the possible presence of insider misuse and/or malware. Code has been discovered for a piece of malware that targets Diebold ATMs (this manufacturer was also a legacy purveyor of voting machines). The code for this malware used undocumented features to

create a virtual ‘skimmer’ capable of recording card details and personal identification numbers without the user’s knowledge, suggesting the creator may have had access to the source code for the ATM. While this does not directly point to an inside job, the possibility certainly cannot be ruled out. Experts at Sophos (a firewall company) believe this code was intended to be preinstalled by an insider at the factory, and would hold transaction details until a special card was entered into the machine—at which point a list of card numbers, PINs, and balances would be printed out for the ne’er-do-well to peruse, and perhaps use, at leisure. It is also possible the malware could be installed by someone with access to the ATM’s internal workings, such as the person who refills the supply of money each day (especially if that malware were to disable or alter the audit process).

Complex Multi-Organizational Systems

One case in which oversight was supposedly provided by corporate approval processes was the disastrous collapse of the Deepwater Horizon. The extraction process in the Gulf of Mexico involved numerous contractors and subcontractors, and all sorts of largely self-imposed monitoring and presumed safety measures. However, as things began to go wrong incrementally, oversight became increasingly complicated—exacerbated further by pressures of contractual time limits and remote managers. This situation is examined in amazing detail in a recent book on this subject.³

Conclusion

Recognition of the risks of systems that are exclusively self-auditing is not new. Although remediations have been repeatedly suggested, the reality is even worse today. We have a much greater dependence on computer- and network-based systems (most of which are riddled with security flaws, potentially subject to external attacks, insider misuse, and denials of service). The technology has not improved with respect to trustworthiness, and the total-system risks have evidently increased significantly.

Independent verification is essential on a spot-check and routine ba-

sis. Security must be designed in, not added on; yet, as we have seen, hacks and exploits can be designed in as well. Hired testers may suffer from tunnel vision based on product objectives or other pressures. Group mentality or fraudulent intent may encourage cover-up of detected failure modes. Whistle-blowers attempting to overcome inadequate self-auditing are often squelched—which tends to suppress reporting. Classified and trade secret systems inherently add to the lack of external oversight.

The bottom line is this: Lacking the ability to independently examine source code (much less recompile it), validate results, and perform spot-checks on deployed devices and system implementations, various anomalies (whether deliberate or unintentional) are very likely to be able to evade detection. Specific questions must be periodically asked and answered, such as: What independent audits are being performed in order to ensure correctness and trustworthiness? When are these audits done? Who is responsible for conducting these audits? Without sufficient and appropriate assurances, self-auditing systems may be nothing more than a charade. 

References

1. Aizuddin, A. The Common Criteria ISO/IEC 15408—The Insight, Some Thoughts, Questions and Issues, 2001; <http://bit.ly/1IVwAr8>
2. Beurdouche, B. et al. A messy state of the union: Taming the composite state machines of TLS. In *Proceedings of the 36th IEEE Symposium on Security and Privacy*, San Jose, CA (May 18–20, 2015); <https://www.smacktls.com/smack.pdf>
3. Boebert, E. and Blossom, J. *Deepwater Horizon: A Systems Analysis of the Macondo Disaster*. Harvard University Press, 2016.
4. Chaum, D. Secret-ballot receipts: True voter-verifiable elections. *IEEE Security and Privacy* 2, 1 (Jan./Feb. 2004).
5. Ewing, J. and Davenport, C. Volkswagen to stop sales of diesel cars involved in recall. *The New York Times* (Sept. 20, 2015).
6. Guttman, B. and Roback, E.A. *An Introduction to Computer Security: The NIST Handbook*. U.S. Department of Commerce, NIST Special Publication 800-12 (Oct. 1995).
7. Mercuri, R. Uncommon criteria. *Commun. ACM* 45, 1 (Jan. 2002).
8. Rako, P. What’s all this meter accuracy stuff, anyhow? *Electronic Design* 16, 41 (Sept. 3, 2013).
9. Thompson, G. et al. In-use emissions testing of light-duty diesel vehicles in the United States. International Council on Clean Transportation (May 30, 2014); <http://www.theicct.org>

Rebecca Mercuri (notable@notablessoftware.com) is a digital forensics and computer security expert who testifies and consults on casework and product certifications.

Peter G. Neumann (neumann@csl.sri.com) is Senior Principal Scientist in the Computer Science Lab at SRI International, and moderator of the ACM Risks Forum.

Copyright held by authors.



DOI:10.1145/2909881

George V. Neville-Neil

Article development led by **acmqueue**
queue.acm.org

Code Vicious

What Are You Trying to Pull?

A single cache miss is more expensive than many instructions.

Dear KV,

I have been reading some pull requests from a developer who has recently been working in code that I also have to look at from time to time. The code he has been submitting is full of strange changes he claims are optimizations. Instead of simply returning a value such as 1, 0, or -1 for error conditions, he allocates a variable and then increments or decrements it, and then jumps to the return statement. I have not bothered to check whether or not this would save instructions, because I know from benchmarking the code those instructions are not where the majority of the function spends its time. He has argued any instruction we do not execute saves us time, and my point is his code is confusing and difficult to read. If he could show a 5% or 10% increase in speed, it might be worth considering, but he has not been able to show that in any type of test. I have blocked several of his commits, but I would prefer to have a usable argument against this type of optimization.

Pull the Other One

Dear Pull,

Saving instructions—how very 1990s of him. It is always nice when people pay attention to details, but sometimes they simply do not pay attention to the right ones. While KV would never encourage developers to waste in-



structions, given the state of modern software, it does seem like someone already has. KV would, as you did, come out on the side of legibility over the saving of a few instructions.

It seems that no matter what advances are made in languages and compilers, there are always programmers who think they are smarter than their tools, and sometimes they are right about that, but mostly they are not. Reading the output of the assembler and counting the instructions may be satisfying for some, but there had better be a lot

more proof than that to justify obfuscating code. I can only imagine a module full of code that looks like this:

```
if (some condition) {
    retval++;
    goto out:
} else {
    retval--;
    goto out:
}
...
out:
    return(retval)
```

and, honestly, I do not really want to. Modern compilers, or even not so modern ones, play all the tricks programmers used to have to play by hand—inlining, loop unrolling, and many others—and yet there are still some programmers who insist on fighting their own tools.

When the choice is between code clarity and minor optimizations, clarity must, nearly always, win. A lack of clarity is the source of bugs, and it is no good having code that is fast and wrong. First the code must be right, then the code must perform; that is the priority that any sane programmer must obey. Insane programmers, well, they are best to be avoided.

The other significant problem with the suggested code is it violates a common coding idiom. All languages, including computer languages, have idioms, as pointed out at length in *The Practice of Programming* by Brian W. Kernighan and Rob Pike (Addison-Wesley Professional, 1999), which I recommended to readers more than a decade ago. Let's not think about the fact the book is still relevant, and that I have been repeating myself every decade. No matter what you think of a computer language, you ought to respect its idioms for the same reason one has to know idioms in a human language—they facilitate communication, which is the true purpose of all languages, programming or otherwise. A language idiom grows organically from the use of a language. Most C programmers, though not all of course, will write an infinite loop in this way:

```
for (;;) {
}
```

or as

```
while (1) {
}
```

with an appropriate break statement somewhere inside to handle exiting the loop when there is an error. In fact, checking the *Practice of Programming* book, I find this is mentioned early on (in section 1.3). For the return case, you mention it is common to return using a value such as 1, 0, or -1 unless the return encodes more than true,

When the choice is between code clarity and minor optimizations, clarity must, nearly always, win.

false, or error. Allocating a stack variable and incrementing or decrementing and adding a `goto` is not an idiom I have ever seen in code, anywhere—and now that you are on the case, I hope I never have to.

Moving from this concrete bit of code to the abstract question of when it makes sense to allow some forms of code trickery into the mix really depends on several factors, but mostly on how much speedup can be derived from twisting the code a bit to match the underlying machine a bit more closely. After all, most of the hand optimizations you see in low-level code, in particular C and its bloated cousin C++, exist because the compiler cannot recognize a good way to map what the programmer wants to do onto the way the underlying machine actually works. Leaving aside the fact that most software engineers really do not know how a computer works, and leaving aside that what most of them were taught—if they were taught—about computers, hails from the 1970s and 1980s before superscalar processors and deep pipelines were a standard feature of CPUs, it is still possible to find ways to speed up by playing tricks on the compiler.

The tricks themselves are not that important to this conversation; what is important is knowing how to measure their effects on the software. This is a difficult and complicated task. It turns out that simply counting instructions as your co-worker has done does not tell you very much about the runtime of the underlying code. In a modern CPU the most precious resource is no longer instructions, except in a very small number of

compute-bound workloads. Modern systems do not choke on instructions; they drown in data. The cache effects of processing data far outweigh the overhead of an extra instruction or two, or 10. A single cache miss is a 32-nanosecond penalty, or about 100 cycles on a 3GHz processor. A simple MOV instruction, which puts a single, constant number into a CPU's register, takes one-quarter of a cycle, according to Agner Fog at the Technical University of Denmark (http://www.agner.org/optimize/instruction_tables.pdf).

That someone has gone so far as to document this for quite a large number of processors is staggering, and those interested in the performance of their optimizations might well lose themselves in that site generally (<http://www.agner.org>).

The point of the matter is that a single cache miss is more expensive than many instructions, so optimizing away a few instructions is not really going to win your software any speed tests. To win speed tests you have to measure the system, see where the bottlenecks are, and clear them if you can. That, though, is a subject for another time.

KV

Related articles on queue.acm.org

Human-KV Interaction

<http://queue.acm.org/detail.cfm?id=1122682>

Quality Software Costs Money—Heartbleed Was Free

Poul-Henning Kamp

<http://queue.acm.org/detail.cfm?id=2636165>

The Network Is Reliable

Peter Bailis and Kyle Kingsbury

<http://queue.acm.org/detail.cfm?id=2655736>

George V. Neville-Neil (kv@acm.org) is the proprietor of Neville-Neil Consulting and co-chair of the ACM Queue editorial board. He works on networking and operating systems code for fun and profit, teaches courses on various programming-related subjects, and encourages your comments, quips, and code snips pertaining to his *Communications* column.

Copyright held by author.



The Profession of IT

How to Produce Innovations

Making innovations happen is surprisingly easy, satisfying, and rewarding if you start small and build up.

YOU HAVE AN idea for something new that could change your company, maybe even the industry. What do you do with your idea?

Promote it through your employer's social network? Put a video about it on YouTube? Propose it on Kickstarter and see if other people are interested? Found a startup? These possibilities have murky futures. Your employer might not be interested, the startup might fail, the video might not go viral, the proposal might not attract followers. And if any of these begins to look viable, it could be several years before you know if your idea is successful. In the face of these uncertainties, it would be easy to give up.

Do not give up so easily. Difficulty getting ideas adopted is a common complaint among professionals. In this column, I discuss why it might not be as difficult as it looks.

The Apparent Weediness of Adoption

Bob Metcalfe's famous story of the Ethernet illustrates the difficulties.² With the provocative title "invention is a flower, innovation is a weed" he articulated the popular impression that creating an idea is glamorous and selling it is grunt work. In his account of Ethernet and the founding of 3Com to sell Ethernets, the invention part happened in 1973–1974 at Xerox PARC. It produced patents, seminal academic papers, and working pro-



totypes. The Ethernet was adopted within Xerox systems. Metcalfe left Xerox in 1979 to found 3Com, which developed and improved the technology and championed it for an international standard (achieved in 1983 as IEEE 802.3). Metcalfe tells of many hours on the road selling Ethernets to executives who had never heard of the technology; he often had only a short time to convince them Ethernet was better than their current local-network technology and they could trust him and his company to deliver. He

did a lot of "down in the weeds" work to get Ethernet adopted.

Metcalfe summarized his experience saying the invention part took two years and the adoption part took 10. He became wealthy not because he published a good paper but because he sold Ethernets for 10 years. He found this work very satisfying and rewarding.

Sense 21

I would like to tell a personal story that sheds light on why adoption might be

rewarding. In 1993, I created a design course for engineers. I called it “Designing a new common sense for engineering in the 21st century,” abbreviated “Sense 21.” The purpose of this course was to show the students how innovation works and how they might be designers who can intentionally produce innovations.

I became interested in doing this after talking to many students and learning about the various breakdowns they had around their aspirations for producing positive change in their organizations and work environments. These students were seniors and graduate students in the age group 20–25. They all were employed by day and took classes in the evening. The breakdowns they discussed with me included: suffering time crunch and information overload, inability to interest people in their ideas, frustration that other “poor” ideas are selected instead of their obviously “better” ideas, belief that good ideas sell themselves, revulsion at the notion you have to sell ideas, complaints that other people do not listen, and complaints that many customers, teammates, and bosses were jerks. I wanted to help these students by giving them tools that would enable them to navigate through these problems instead of being trapped by them. I created the Sense 21 course for them.

I announced to the students that the course outcome is “produce an innovation.” That meant each of them would find an innovation opportunity and make it happen. To get there we would need to understand what innovation is—so we can know what we are to produce—and to learn some foundational tools of communication that are vital for making it happen.

We spent the first month learning the basics of generating action in language—specifically speech acts and the commitments they generate, and how those commitments shape their worlds.¹ There is no action without a commitment, and commitments are made in conversations. The speech acts are the basic moves for making commitments. What makes this so fundamental is there are only five kinds of commitments (and speech acts) and therefore the basic communication

The alternative sense of language as generator and shaper gave rise to a new definition of automation.

tools are simple, universal, and powerful. With this we were challenging the common sense that the main purpose of language is to communicate messages and stories. We were after a new sense: with language we make and shape the world.

Everett Rogers, whose work on innovation has been very influential since 1962, believed communication was essential to innovation. Paraphrasing Rogers: “Innovation is the creation of a novel proposal that diffuses through the communication channels of a social network and attracts individuals to decide to adopt the proposal.”³

The message sense of communication permeates this view: an innovation proposal is an *articulation and description* of a novel idea to solve a problem, and adoption is an individual *decision made after receiving messages* about the proposal.

My students struggled with this definition of innovation. They could not see their own agency in adoption. How do they find and articulate novel ideas? What messages should they send, over which channels? How do they find and access existing channels? Should they bring the message to prospective adopters by commercials, email, or personal visits? What forms of messages are most likely to influence a positive decision? How do they deal with the markedly different kinds of receptivity to messages among early, majority, and laggard adopters? Should they be doing something else altogether? The definition gave no good answers for such questions.

The alternative sense of language as generator and shaper gave rise to a new definition of innovation, which we used in the course: “Innovation is adoption

Calendar of Events

June 2–4

SIGMIS-CPR ‘16: 2015 Computers and People Research Conference, Washington, D.C., Sponsored: ACM/SIG, Contact: Jeria Quesenberry, Email: jqquesenberry@cmu.edu

June 4–8

DIS ‘16: Designing Interactive Systems Conference 2016, Brisbane, QLD, Australia, Sponsored: ACM/SIG, Contact: Marcus Foth, Email: marcus.foth@gmail.com

June 8–10

PASC ‘16: Platform for Advanced Scientific Computing Conference, Lausanne, Switzerland, Contact: Olaf Schenk, Email: olaf.schenk@usi.ch

June 14–18

SIGMETRICS ‘16: SIGMETRICS/ PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems, Antibes, Juan-Les-Pins, France, Contact: Sara Alouf, Email: Sara.Alouf@inria.fr

June 18–22

ISCA ‘16: The 42nd Annual International Symposium on Computer Architecture, Seoul, Republic of Korea, Contact: Gabriel Loh, Email: gabriel.loh@amd.com

June 19–23

JCDL ‘16: The 16th ACM/IEEE-CS Joint Conference on Digital Libraries, Newark, NJ, Contact: Lillian N. Cassel, Email: lillian.cassel@villanova.edu

June 20–22

PerDis ‘16: The International Symposium on Pervasive Displays, Oulu, Finland, Sponsored: ACM/SIG, Contact: Vassilis Kostakos, Email: vassilis.spam@kostakos.org

of new practice in a community, displacing other existing practices.”

The term “practice” refers to routines, conventions, habits, and other ways of doing things, shared among the members of a community. Practices are embodied, meaning people perform them without being aware they are exercising a skill. Technologies are important because they are tools that enable and support practices. Since people are always doing something, adopting a new practice means giving up an older one. This was the most demanding of all the definitions of innovation. It gives an acid test of whether an innovation has happened.

With this formulation, student questions shifted. Who is my community? How do I tell whether my proposal will interest them? Is training with a new tool a form of adoption? Who has power to help or resist? Their questions shifted from forms of communication to how they could engage with their community. A summary answer to their engagement questions was this process:

- ▶ Listen for concerns and breakdowns in your community
- ▶ Gather a small team
- ▶ Design and offer a new tool—a combination or adaptation of existing technologies that addresses the concern
- ▶ Mobilize your community into using the tool
- ▶ Assess how satisfied they are with their new practice

To start their project, I asked them to find a small group of about five people in their work environment. This group would be their innovating community. I did not impose much structure on them because I wanted them to learn how to navigate the unruly world they would be discovering. I asked them to give progress reports to the whole group, who frequently gave them valuable feedback and gave me opportunities to coach the group.

Here is an example of an incident that helped a student—let’s call him “Michael”—learn to listen for concerns. Michael was unhappy with me because I declined his request to let him use workstations in my lab for a project unrelated to the lab. In class, I asked Michael to repeat his request. He did so enthusiastically and quickly fell into a confrontational mood.

Could it be that finding out the concerns of your community might be as simple as asking “What are your concerns?”

He tried half a dozen different arguments on me, all variations on the theme that I was acting unethically or irrationally in denying his request. None moved me. Soon the entire class was offering suggestions to Michael. None of that moved me either. After about 10 minutes, Michael hissed, “Are you just playing with me? Saying no just for spite? What’s wrong with my request? It’s perfectly reasonable!” I said, “You have not addressed any of my concerns.” With utter frustration, he threw his hands into the air and exclaimed, “But I don’t even know what you are concerned about!” I smiled at him, leaned forward, and said, “Exactly.”

Convulsed by a Great Aha!, Michael turned bright red and proclaimed, “Geez, now I get what you mean by listening.” The other members of the class looked startled and got it too. Then they excitedly urged him on: “Ask him what he is concerned about!” This he did. Soon he proposed to fashion his project to help contribute to the goals of the lab. I was seduced. We closed a deal.

Could it be that finding out the concerns of your community might be as simple as asking “What are your concerns?”

By the end of the semester we had worked our way through the stages of the process and coached on other fine points. They all succeeded in producing an innovation. In our final debriefing they proclaimed an important discovery: innovations do not have to be big.

That was very important. The innovation stories they had learned all their lives told them all innovations

are big world-shakers and are the work of geniuses. However, their own experiences told them they could produce small innovations even if they were not geniuses. Moreover, they saw they could increase the size of their innovation communities over time as they gained experience.

Getting It Done

A cursory reading of the Metcalfe story could lead you to conclude the full Ethernet innovation took 10 years. That is a very long time. If you believe you will not see the fruits of your work for 10 years, you are unlikely to undertake the work. If on the other hand you believe your work consists of an ongoing series of small innovations, you will find your work enjoyable, and after 10 years you will find it has added up to a large innovation. This is what Metcalfe wanted to tell us. He enjoyed his work and found that each encounter with a new company that adopted Ethernet was a new success and a new small innovation.

The students said one other thing that startled me. They said that taking the course and doing the project was life altering for them. The reason was the basic tools had enabled them to be much more effective in generating action through all parts of their lives. The realization that we generate action through our language is extraordinarily powerful.

If we can tell the stories and satisfying experiences of innovators doing everyday, small innovation, we will have a new way to tell the innovation story and lead people to more success with their own innovations. Innovation is no ugly weed. Like a big garden of small flowers, innovation is beautiful. **C**

References

1. Flores, F. *Conversations for Action and Collected Essays*. CreateSpace Independent Publishing Platform, 2013.
2. Metcalfe, R. Invention is a flower, innovation is a weed. *MIT Technology Review* (Nov. 1999); <http://www.technologyreview.com/featuredstory/400489/invention-is-a-flower-innovation-is-a-weed/>
3. Rogers, E. *Diffusion of Innovations* (5th ed. 2003). Free Press, 1962.

Peter J. Denning (pjd@nps.edu) is Distinguished Professor of Computer Science and Director of the Cebrowski Institute for information innovation at the Naval Postgraduate School in Monterey, CA, is Editor of *ACM Ubiquity*, and is a past president of ACM. The author’s views expressed here are not necessarily those of his employer or the U.S. federal government.

Copyright held by author.

Interview

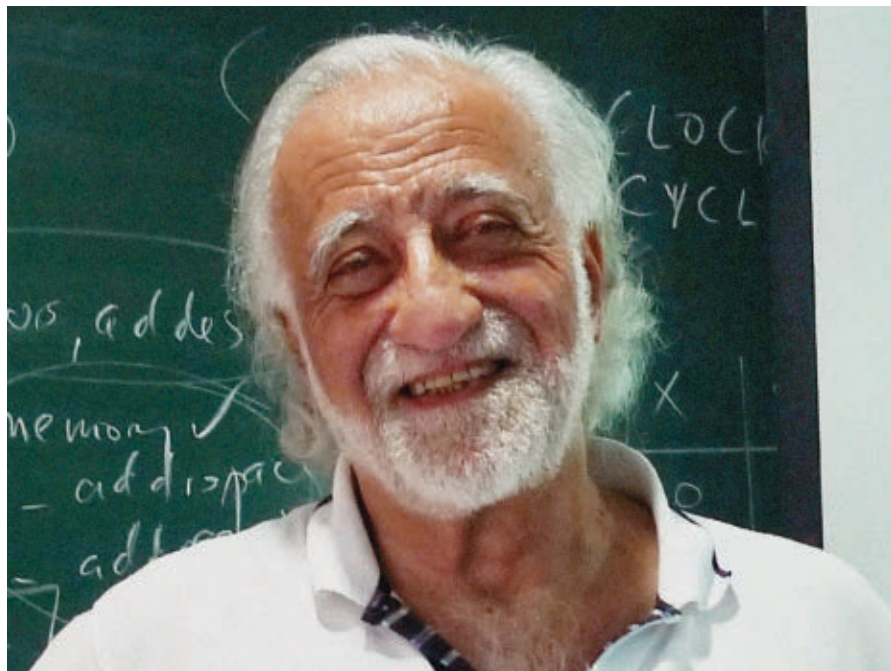
An Interview with Yale Patt

ACM Fellow Professor Yale Patt reflects on his career in industry and academia.

PROFESSOR YALE PATT, the Ernest Cockrell, Jr. Centennial Chair in Engineering at The University of Texas at Austin has been named the 2016 recipient of the Benjamin Franklin Medal in Computer and Cognitive Science by the Franklin Institute. Patt is a renowned computer architect, whose research has resulted in transformational changes to the nature of high-performance microprocessors, including the first complex logic gate implemented on a single piece of silicon. He has received ACM's highest honors both in computer architecture (the 1996 Eckert-Mauchly Award) and in education (the 2000 Karl V. Karlstrom Award). He is a Fellow of the ACM and the IEEE and a member of the National Academy of Engineering.

Derek Chiou, an associate professor of Electrical and Computer Engineering at The University of Texas at Austin, conducted an extensive interview of Patt, covering his formative years to his Ph.D. in 1966, his career since then, and his views on a number of issues. Presented here are excerpts from that interview; the full interview is available via the link appearing on the last page of this interview.

DEREK CHIOU: Let's start with the influences that helped shape you into who you are. I have often heard you comment on your actions as, "That's the way my mother raised me." Can you elaborate?



Yale Patt, ACM Fellow and Ernest Cockrell, Jr. Centennial Chair Professor at The University of Texas at Austin.

YALE PATT: In my view my mother was the most incredible human being who ever lived. Born in Eastern Europe, with her parents' permission, at the age of 20, she came to America by herself. A poor immigrant, she met and married my father, also from a poor immigrant family, and they raised three children. We grew up in one of the poorer sections of Boston. Because of my mother's insistence, I was the first from that neighborhood to go to college. My brother was the second. My sister was the third.

You have often said that as far as your professional life is concerned, she taught you three important lessons.

That is absolutely correct. Almost everyone in our neighborhood quit school when they turned 16 and went to work in the Converse Rubber factory, which was maybe 100 yards from our apartment. She would have none of it. She knew that in America the road to success was education. She insisted that we stay in school and that we achieve. An A-minus was not acceptable. "Be the best that you can be."

That was the first lesson. The second lesson: “Once you do achieve, your job is to protect those who don’t have the ability to protect themselves.” And I have spent my life trying to do that. The third lesson is to not be afraid to take a stand that goes against the currents—to do what you think is right regardless of the flak you take. And I have certainly taken plenty of flak. Those were the three lessons that I believe made me into who I am. When I say that’s the way my mother raised me, it usually has to do with one of those three principles.

What about your father?

My father was also influential—but in a much quieter way. We didn’t have much money. It didn’t matter. He still took us to the zoo. He took us to the beach. He took me to my first baseball game. He got me my first library card—taught me how to read. I remember us going to the library and getting my first library card at the age of five. So when I started school, I already knew how to read. That was my father’s influence.

I understand there is a story about your father that involves your first marathon.

Yes, the New York City Marathon. The first time I ran it was in 1986. If you finish, they give you a medal. I gave it to my father. “Dad, this is for you.” He says, “What’s this?” I said, “It’s a medal.” “What for?” “New York City Marathon.” “You won the New York City Marathon?” “No, Dad. They give you a medal if you finish the New York City Marathon.” And then he looked at me in disbelief. “You mean you lost the New York City Marathon?” It was like he had raised a loser, and I realized that he too, in his quieter way, was also pushing me to achieve and to succeed.

Besides your parents there were other influences. For example, you’ve often said Bill Linvill was the professor who taught you how to be a professor.

Bill Linvill was incredible. He was absolutely the professor who taught me how to be a professor—that it’s not about the professor, it’s about the students. When he formed the new Department of Engineering Economic Systems, I asked if I could join him. “No way,” he said. “You are a qualified Ph.D. candidate in EE. You will get your Ph.D. in EE, and that will open lots of

doors for you. If you join me now, you will be throwing all that away, and I will not let you do that. After you graduate, if you still want to, I would love to have you.” That was Bill Linvill. Do what is best for the students, not what is best for Bill Linvill.

You did your undergraduate work at Northeastern. Why Northeastern?

Northeastern was the only school I could afford financially, because of the co-op plan. Ten weeks of school, then ten weeks of work. It was a great way to put oneself through engineering school.

What do you think of co-op now?

I think it’s an outstanding way to get an education. The combination of what I learned in school and what I learned on the job went a long way toward developing me as an engineer. In fact, I use that model with my Ph.D. students. Until they are ready to devote themselves full time to actually writing the dissertation, I prefer to have them spend their summers in industry. I make sure the internships are meaningful, so when they return to campus in the fall, they are worth a lot more than when they left at the beginning of the summer. The combination of what we can teach them on campus and what they can learn in industry produces Ph.D.’s who are in great demand when they finish.

I understand you almost dropped out of engineering right after your first engineering exam as a sophomore.

Yes, the freshman year was physics, math, chemistry, English, so my first engineering course came as a sophomore. I did so badly on my first exam I wasn’t even going to go back and see just how badly. My buddies convinced me we should at least go to class and find out. There were three problems on the exam. I knew I got one of them. But one of them I didn’t even touch, and the third one I attempted, but with not great success. It turns out I made a 40. The one I solved I got 33 points for. The one I didn’t touch I got 0. And the one I tried and failed I got seven points. The professor announced that everything above a 25 was an A. I couldn’t believe it. In fact, it took me awhile before I understood.

Engineering is about solving problems. You get no points for repeating what the professor put on the blackboard. The professor gives you problems you have not seen before. They have taught you what you need to solve them. It is up to you to show you can. You are not expected to get a 100, but you are expected to demonstrate you can think and can crack a problem that you had not seen before. That’s what engineering education is about.

Then you went to Stanford University for graduate work. Why did you choose Stanford?

My coop job at Northeastern was in microwaves, so it seemed a natural thing to do in graduate school. And, Stanford had the best program in electromagnetics.

But you ended up in computer engineering. How did that happen?

There’s a good example of how one professor can make a difference. At Stanford, in addition to your specialty, they required that you take a course in some other part of electrical engineering. I chose switching theory, which at the time we thought was fundamental to designing computers, and we recognized computers would be important in the future. The instructor was a young assistant professor named Don Epley. Epley really cared about students, made the class exciting, made the class challenging, was always excited to teach us and share what he knew. By the end of the quarter, I had shifted my program to computers and never looked back.

The rumor is you wrote your Ph.D. thesis in one day. What was that all about?

Not quite. I made the major breakthrough in one day. As you know, when you are doing research, at the end of each day, you probably don’t have a lot to show for all you did that day. But you keep trying. I was having a dry spell and nothing was working. But I kept trying. I had lunch, and then I’d gone back to my cubicle. It was maybe 2:00 in the afternoon. All of a sudden, everything I tried worked. The more I tried, the more it worked. I’m coming up with algorithms, and I’m proving theorems. And it’s all coming together, and, my heart is racing at this point. In fact, that’s what makes research worth-

while—those (not often) moments when you’ve captured new knowledge, and you’ve shown what nobody else has been able to show. It’s an amazing feeling. Finally I closed the loop and put the pen down. I was exhausted; it was noon the next day. I had worked from 2:00 in the afternoon all the way through the night until noon the next day, and there it was. I had a thesis!

So you wrote your thesis in one day.

No, I made the breakthrough in one day, which would not have happened if it had not been for all those other days when I kept coming up empty.

What did you do then?

I walked into my professor’s office. He looked up from his work. I went to the blackboard, picked up the chalk, and started writing. I wrote for two hours straight, put down the chalk and just looked at him. He said, “Write it up and I’ll sign it. You’re done.”

After your Ph.D., your first job was as an assistant professor at Cornell University. Did you always plan on teaching?

No. I always thought: Those who can do; those who can’t, teach. I interviewed with 10 companies, and had nine offers. I was in the process of deciding when Fred Jelinek, a professor at Cornell, came into my cubicle and said, “We want to interview you at Cornell.” I said, “I don’t want to teach.” He said, “Come interview. Maybe you’ll change your mind.” So there I was, this poor boy from the slums of Boston who could not have gotten into Cornell back then, being invited to maybe teach there. I couldn’t turn down the opportunity to interview, so I interviewed, and I was impressed—Cornell is an excellent school.

Now I had 10 offers. After a lot of agonizing, I decided on Cornell. All my friends said, “We knew you were going to decide on Cornell because that’s what you should be—a teacher.” And they were right! I was very lucky. If Fred Jelinek had not stumbled into my cubicle, I may never have become a professor, and for me, it’s absolutely the most fantastic way to go through life.

Why did you only spend a year there?

At the time, the U.S. was fighting a war in Vietnam. I was ordered to report

“The combination of what I learned in school and what I learned on the job went a long way toward developing me as an engineer.”

to active duty in June 1967, at the end of my first year at Cornell. I actually volunteered; I just didn’t know when my number would come up.

Your active duty started with boot camp. What was that like?

Boot camp was amazing. Not that I would want to do it again, but I am glad I did it once. It taught me a lot about the human spirit, and the capabilities of the human body that you can draw on if you have to.

What happened after boot camp?

After nine weeks of boot camp, I was assigned to the Army Research Office for the rest of my two-year commitment. I was the program manager for a new basic research program in computer science. I was also the Army’s representative on a small committee that was just beginning the implementation of the initial four-node ARPANET. I knew nothing about communication theory, but I had a Ph.D. in EE, and had been a professor at Cornell, so someone thought I might be useful. In fact, it was an incredible learning experience. I had fantastic tutors: Lenny Kleinrock and Glen Culler. Lenny had enormous critical expertise in both packet switching and queueing theory. Glen was a professor at UC Santa Barbara, trained as a mathematician, but one of the best engineers I ever met. In fact, I give him a lot of the credit for actually hacking code and getting the initial network to work.

After the Army, you stayed in North Carolina, taught at NC State, then moved to San Francisco State to build

their computer science program. Then you went to Berkeley. You were a visiting professor at Berkeley from 1979 to 1988. What was that like?

Berkeley was an incredible place at that time. Mike Stonebraker was doing Ingres, Sue Graham had a strong compiler group, Dick Karp and Manny Blum were doing theory, Domenico Ferrari was doing distributed UNIX, Velvel Kahan was doing IEEE Floating Point, Dave Patterson with Carlo Sequin had started the RISC project, and I and my three Ph.D. students Wen-mei Hwu, Mike Shebanow, and Steve Melvin were doing HPS. In fact, that is where HPS was born. We invented the Restricted Data Flow model, showed that you could do out-of-order execution and still maintain precise exceptions, and that you could break down complex instructions into micro-ops that could be scheduled automatically when their dependencies were resolved. We had not yet come up with the needed aggressive branch predictor, but we did lay a foundation for almost all the cutting-edge, high-performance microprocessors that followed.

You had other Ph.D. students at Berkeley as well.

Yes, I graduated six Ph.D.’s while I was at Berkeley—I guess a little unusual for a visiting professor. The other three were John Swensen, Ashok Singhal, and Chien Chen. John was into numerical methods and showed that an optimal register set should contain a couple of very fast registers when latency is the critical issue and a large number of slow registers when throughput is critical. Ashok and Chien worked on implementing Prolog, which was the focal point of the Aquarius Project, a DARPA project that Al Despain and I did together.

Then you went to Michigan. Two things stand out at Michigan: first, your research in branch prediction.

We actually did a lot of research in branch prediction during my 10 years at Michigan, but you are undoubtedly thinking of our first work, which I did with my student Tse-Yu Yeh. Tse-Yu had just spent the summer of 1990 working for Mike Shebanow at Motorola. Mike was one of my original

HPS students at Berkeley. When Tse-Yu returned to Michigan at the end of the summer, he had some ideas about branch prediction, based on his interaction with Shebanow. He and I ended up with the two-level adaptive branch predictor which we published in *Micro* in 1991. Intel was the first company to use it. When they moved from a five-stage pipeline on Pentium to a 12-stage pipeline on Pentium Pro, they could not afford the misprediction penalty they would have gotten with their Pentium branch predictor. So, they adapted ours. Since then, some variation has been used by just about everybody.

Michigan is also where you developed the freshman course.

Yes, I had wanted to teach that material to freshmen for a long time, but always ran up against a brick wall. Then in 1993, the faculty were complaining that students didn't understand pointer variables and recursion was magic. I just blurted out, "The reason they don't understand is they have no idea what's going on underneath. If we really want them to understand, then we have to start with how the computer works." I offered to do it, and the faculty said okay. Kevin Compton and I developed the freshman course, and in fall 1995, we taught it for the first time. In fall 1996, it became the required first course in computing, and we taught it to all 400 EECS freshmen.

I heard Trevor Mudge volunteered to teach it if something happened.

Trevor said he would be willing to teach the course if we gave him a book. There was no book. In fact, the course was completely different from every freshman book on the market. We started with the transistor as a wall switch. Kids have been doing wall switches since they were two years old, so it was not difficult to teach them the switch level behavior of a transistor. From wall switches we made inverters, and then NAND gates and NOR gates, followed by muxes and decoders and latches and memory, then a finite state machine, and finally a computer. They internalized the computer, bottom-up, and then wrote their first program in the machine language of the LC-2,

"Engineering is about solving problems. You get no points for repeating what the professor put on the blackboard."

a computer I invented for the course. Programming in 0s and 1s gets old very quickly, so we quickly moved to LC-2 assembly language.

Since Trevor needed a textbook to teach the course in the spring, I wrote the first draft over Christmas vacation. That's why the freshman textbook was born. If Trevor hadn't insisted, who knows? There may not have been a freshman textbook. But there was no other book available because it was a complete departure from everybody else.

You ended up co-authoring the book with one of your Ph.D. students.

Yes, originally, it was going to be with Kevin Compton, but Kevin ended up not having time to do it. So I asked Sanjay Patel, one of my Ph.D. students who TA'd the course the first year we offered it. We wrote the book together, and published it as he was finishing his Ph.D.

You left Michigan in 1999 to come to Texas. Is there anything at Texas that particularly stands out?

Far and away, my students and my colleagues. I have now graduated 12 Ph.D.'s at Texas. When I came here, I brought my Michigan Ph.D. students with me. Two of them, Rob Chappell and Paul Racunas, received Michigan degrees but actually finished their research with me at UT. Two others, Mary Brown and Francis Tseng, were early enough in the Ph.D. program that it made more sense for them to transfer. Mary graduated from UT in 2005, went to IBM, rose to be one of the key architects of their Power 8 and 9 chips, and

recently left IBM to join Apple. Francis got his Ph.D. in 2007, and joined Intel's design center in Hillsboro, Oregon.

With respect to my colleagues, I consider one of my biggest achievements that I was able to convince you and Mattan Erez to come to Texas. The two of you are, in a major way, responsible for building what we've got in the computer architecture group at Texas.

Six of your students are professors?

That's right. Three of them hold endowed chairs. Wen-Mei Hwu is the Sanders Chair at Illinois. Greg Ganger, one of my Michigan Ph.D.'s, holds the Jatras Chair at Carnegie Mellon, and Onur Mutlu, one of my Texas Ph.D.'s holds the Strecker chair at Carnegie-Mellon. In total, I have two at Illinois, Wen-Mei Hwu and Sanjay Patel, also a tenured full professor, two at Carnegie Mellon, Greg Ganger and Onur Mutlu, and two at Georgia Tech, Moin Qureshi, and Hyesoon Kim, both associate professors.

And a number of your students are doing great in industry too.

Yes. I already mentioned Mary Brown. Mike Shebanow has designed a number of chips over the years, including the Denali chip at HAL and the M1 at Cyrix. He was also one of the lead architects of the Fermi chip at Nvidia. Mike Butler, my first Michigan Ph.D., was responsible for the bulldozer core at AMD. Several of my students play key roles at Intel and Nvidia.

You are well known for speaking your mind on issues you care about, and have some very strong views on many things. Let's start with how you feel about the United States of America.

Quite simply, I love my country. I already mentioned that I spent two years in the Army—voluntarily. I believe everyone in the U.S. should do two years of service, and that nobody should be exempt. It's not about letting the other guy do it. It's about every one of us accepting this obligation. I believe in universal service. It does not have to be the military. It can be the Peace Corps, or Teach for America, or some other form of service.

I also believe in immigration. That's another key issue in the U.S. today. Immigration is part of the core of the

American fabric. It has contributed enormously to the greatness of America. Some people forget that unless you're a Native American we all come from immigrant stock. The Statue of Liberty says it well: "Give me your tired, your poor." It is a core value of America. I hope we never lose it.

I also believe in the Declaration of Independence as the founding document of America, and the Constitution as the codification of that document. Most important are the 10 amendments Jefferson put forward that represent the essence of America. "We hold these truths to be self-evident," that some rights are too important to leave to the will of the majority, that they are fundamental to every human being. And that's also come under siege lately. Freedom of speech, assembly, free from unlawful search and seizure, habeas corpus, the knowledge that the police can't come and pick you up and lock you up and throw the key away. Some of this seems to have gotten lost over the last few years. I remain hopeful we will return to these core values, that nothing should stand in the way of the first 10 amendments to the Constitution.

Let's talk about your research and teaching. Can you say something about how you mentor your Ph.D. students in their research?

I don't believe in carving out a problem and saying to the student, "Here's your problem. Turn the crank; solve the problem." I have a two-hour meeting every week with all my graduate students. My junior students are in the room when I push back against my senior students. Initially, they are assisting my senior students so they can follow the discussion. At some point, they identify a problem they want to work on. Maybe during one of our meetings, maybe during a summer internship, whenever. I encourage them to work on the problem. They come up with stuff, and I push back. If they get too far down a rat hole, I pull them back. But I cut them a lot of slack as I let them continue to try things. In most cases, eventually they do succeed.

Don't research-funding agencies require you to do specific kinds of research?

I don't write proposals to funding

agencies. I've been lucky that my research has been supported by companies. It is true that in this current economy, money is harder to get from companies. So if any companies are reading this and would like to contribute to my research program and fund my Ph.D. students, I'll gladly accept a check. The checks from companies come as gifts, which means there is no predetermined path we are forced to travel; no deliverables we have promised. In fact, when we discover we are on the wrong path, which often happens, we can leave it. My funding has come almost exclusively from companies over the last 40 years so I don't have that problem.

There is a story about you wanting to give your students a shovel.

As I have already pointed out, most days nothing you try works out so when it is time to call it a day, you have nothing to show for all your work. So I've often thought what I should do is give my student a shovel and take him out in the backyard and say, "Dig a hole." And he would dig a hole. And I'd say, "See? You've accomplished something. You can see the hole you've dug." Because at the end of most days, you don't see anything else.

The next day, the student still doesn't see anything, so we go to the backyard again. "Now fill in the hole." So, again, he could see the results of what he did. And that's the way research goes day after day, until you make the breakthrough. All those days of no results provides the preparation so that when the idea hits you, you can run with it. And that's when the heart pounds. There is nothing like it. You've uncovered new knowledge.

Can you say something about your love for teaching?

It's the thing I love most. These kids come in, and I'm able to make a difference, to develop their foundation, to see the light go on in their eyes as they understand difficult concepts. In my classroom, I don't cover the material. That's their job. My job is to explain the tough things they can't get by themselves. I entertain questions. Even in my freshman class with 400 students, I get questions all the time. Some people say lectures are bad. Bad lectures are

bad. My lectures are interactive—I'm explaining the tough nuts, and the students ask questions. And they learn.

I know you have a particular dislike for lip service instead of being real.

Being real is very important. The kids can tell whether you're spouting politically correct garbage or whether you're speaking from the depths of your soul. If you're real with them, they will cut you enormous slack so you can be politically incorrect and it doesn't matter to them because they know you're not mean spirited. They know you're real. And that's what's important.

What do you think about Texas' seven-percent law that forces the universities to admit the student if he's in the top seven percent of the high school graduating class, since many of them are really not ready for the freshman courses?

It is important to provide equal opportunity. In fact, my classroom is all about equal opportunity. I don't care what race, I don't care what religion, I don't care what gender. I welcome all students into my classroom and I try to teach them. The seven-percent law admits students who come from neighborhoods where they didn't get a proper high school preparation. And this isn't just the black or Hispanic ghettos of Houston. It's also rural Texas where white kids don't get the proper preparation. It's for anyone who is at the top of the class, but has not been prepared properly. The fact they're in the top of the class means they're probably bright. So we should give them a chance. That's what equal opportunity is all about—providing the chance. The problem is that when we welcome them to the freshman class, we then tell them we want them to graduate in four years. And that's a serious mistake because many aren't yet ready for our freshman courses. They shouldn't be put in our freshman courses.

If we're serious about providing equal opportunity for these students, then we should provide the courses to make up for their lack of preparation, and get them ready to take our freshman courses. And if that means it takes a student more than four years to graduate, then it takes more than four years

to graduate. I don't care what they know coming in. What I care about is what they know when they graduate. At that point I want them to be every bit as good as the kids who came from the best highly prepared K-12 schools. We can do that if we're willing to offer the courses to get them ready for our freshman courses.

Can you say something about your Ten Commandments for good teaching?

On my website I have my Ten Commandments. For example, memorization is bad. The students in my freshman course have been rewarded all through school for their ability to memorize, whether or not they understood anything. And now they are freshman engineering students expecting to succeed by memorizing. But engineering is about thinking and problem solving, not memorizing. So I have to break them of that habit of memorizing.

There are other commandments. You should want to be in the classroom. You should know the material. You should not be afraid of interruptions. If I explain them all, this interview will go on for another two or three hours, so I should probably stop. If you want to see my Ten Commandments, they're on my website.^a

There was an incident regarding your younger sister in a plane geometry course. What was that about?

That was a perfect example of memorization. I was visiting my parents, and my sister, who was studying plane geometry at the time, asked me to look at a proof that had been marked wrong on her exam paper. Her proof was completely correct. All of a sudden it hit me! I had gone to the same high school and in fact had the same math teacher. He absolutely did not understand geometry. But he was assigned to teach it. So what did he do? This was before PowerPoint. The night before, he would copy the proof from the textbook onto a sheet of paper. In class he would copy the proof onto the blackboard. The students would copy the proof into their notes. The night before the exam, they'd memorize the

proof. On the exam he'd ask them to prove what he had put on the board. They had no idea what they were doing, but they'd memorized the proof. The result: 100% on the exam.

My sister didn't memorize proofs. She understood plane geometry. She read the theorem, and came up with a proof. It's not the proof that was in the book. But as you well know, there are many ways to prove a theorem. The teacher did not understand enough geometry to be able to recognize that even though her proof was not the proof in the book, her proof was correct. So she got a zero! Memorization!

You once told me about a colleague at Michigan who came into your office one day after class complaining he had given the worst lecture of his life.

Yes, a very senior professor. He came into my office, slammed down his sheaf of papers, "I've just given the worst lecture of my life. I'm starting my lecture, and I've got 10 pages of notes I need to get through. I get about halfway through the first page, a kid asks a question. And I think, this kid hasn't understood anything. So I made the mistake of asking the class, who else doesn't understand this? Eighty percent of their hands go up. I figured there's no point going through the remaining 9½ pages if they don't understand this basic concept. I put my notes aside, and spent the rest of the hour teaching them what they needed to understand in order for me to give today's lecture. At the end of the lecture, I've covered nothing that I had planned to cover because I spent all the time getting the students ready for today's lecture. The worst day of my life."

I said, "Wrong! The best day of your life. You probably gave them the best lecture of the semester." He said, "But I didn't cover the material." I said, "Your job is to explain the hard things so they can cover the material for themselves." He adopted this approach, and from then on, he would check regularly. And if they didn't understand, he would explain. He never got through all the material.

In fact, that's another one of my Ten Commandments. Don't worry about getting through all the material. Make sure you get through the core mate-

rial, but that's usually easy to do. The problem is that back in August when you're laying out the syllabus, you figure every lecture will be brilliant, every kid will come to class wide awake, ready to learn, so everything will be fine. Then the semester begins. Reality sets in. Not all of your lectures are great. It's a reality. Not all kids come to class wide awake. It's a reality. So you can't get through everything you thought you would back in August. But you can get through the core material. So don't worry about getting through everything. And don't be afraid to be interrupted with questions. He adopted those commandments and ended up with the best teaching evaluations he had ever received.

You got your Ph.D. 50 years ago. Your ideas have made major impact on how we implement microprocessors. Your students are endowed chairs at top universities. Your students are at the top of their fields in the companies where they work. You've won just about every award there is. Isn't it time to retire?

Why would I want to retire? I love what I'm doing. I love the interaction with my graduate students in research. I enjoy consulting for companies on microarchitecture issues. Most of all, I love teaching. I get to walk into a classroom, and explain some difficult concept, and the kids learn, the lights go on in their eyes. It's fantastic. Why would I want to retire? I have been doing this now, for almost 50 years? I say I am at my mid-career point. I hope to be doing it for another 50 years. I probably won't get to do it for another 50 years. But as long as my brain is working and as long as I'm excited about walking into a classroom and teaching, I have no desire to retire. ■

Derek Chiou (derek@utexas.edu) is an associate professor of Electrical and Computer Engineering at The University of Texas at Austin and a partner hardware architect at Microsoft Corporation.



Watch the authors discuss their work in this exclusive *Communications* video.
<http://cacm.acm.org/videos/an-interview-with-yale-patt>

For the full-length video, please visit <https://vimeo.com/an-interview-with-yale-patt>

a <http://users.ece.utexas.edu/~patt/Ten.commandments>

Viewpoint

Computer Science Should Stay Young

Seeking to improve computer science publication culture while retaining the best aspects of the conference and journal publication processes.

UNLIKE MOST OTHER academic fields, refereed conferences in computer science are generally the most prestigious publication venues. Some people have argued computer science should “grow up” and adopt journals as the main venue of publication, and that chairs and deans should base hiring and promotion decisions on candidate’s journal publication record as opposed to conference publications.^{a,b}

While I share a lot of the sentiments and goals of the people critical of our publication culture, I disagree with the conclusion that we should transition to a classical journal-based model similar to that of other fields. I believe conferences offer a number of unique advantages that have helped make computer science dynamic and successful, and can continue to do so in the future.

First, let us acknowledge that no peer-review publication system is perfect. Reviewers are inherently subjective and fallible, and the amount of papers being written is too large to allow as careful and thorough review of each submission as should ideally be the case. Indeed, I agree with many of the critiques leveled at computer science conferences, but also think these critiques could apply equally well to



any other peer-reviewed publication system. That said, there are several reasons I prefer conferences to journals:

► **A talk is more informative than a paper.** At least in my area (theory), I find I can get the main ideas of a piece of work much better by hearing a talk about it than by reading the paper. The written form can be crucial when you really need to know all the details, but a talk is better at conveying the high-order bits that most of us care about. I think that our “conference first” culture in computer science has resulted with much better talks (on average) than those of many journal-focused disciplines.

► **Deadlines make for more efficient reviewing.** As an editor for the *Journal of the ACM*, I spend much time chasing down potential reviewers for every submission. At this rate, it would have taken me decades to process the amount of papers I handled in six months as the program chair of the FOCS confer-

ence. In a conference you line up a set of highly qualified reviewers (that is, the program committee) ahead of the deadline, which greatly reduces the administrative overhead per submission.

► People often lament the quality of reviews done under time pressure, but no matter how we organize our refereeing process, if X papers are being written each year, and the community is willing to dedicate Y hours to review them in total, on average a paper will always get Y/X hours of reviewer attention. I have yet to hear a complaint from a reviewer that they would have liked to spend a larger fraction of their time refereeing papers, but have not been able to do so due to the tight conference schedule. Thus, I do not expect an increase in Y if journals were to suddenly become our main avenue of publication. If this happened, then journals would have the same total refereeing resources to deal with the same mass of submissions conferences currently do and it is unrealistic to expect review quality would be magically higher.

► **Conferences have rotating gatekeepers.** A conference program committee typically changes at every iteration, and often contains young people such as junior faculty or postdocs that have a unique perspective and are intimately familiar with cutting-edge research. In contrast, editorial boards of journals are much more stable and senior. This can sometimes be a good thing but also poses the danger of keeping out great works that are not appeal-

a Moshe Vardi, Editor’s letter, *Communications* (May 2009); <http://bit.ly/1UngC33>

b Lance Fortnow, “Time for Computer Science to Grow Up,” *Communications* (Aug. 2009); <http://bit.ly/1XQ6RrW>

ing to the particular board members. Of course, one could imagine a journal with a rotating board, but I think there is a reason this configuration works better at a conference. It is much easier for program committee members to judge papers in batch, comparing them with one another, than to judge each paper in isolation as they would in a journal. This holds doubly so for junior members, who cannot rely on extensive experience when looking at individual papers, and who benefit greatly from the highly interactive nature of the conference decision process.

Related to the last point, it is worthwhile to mention the NIPS 2014 experiment, where the program chairs, Corinna Cortes and Neil Lawrence, ran a duplicate refereeing process for 10% of the submissions, to measure the agreement in the accept/reject decisions. The overall agreement was roughly 74% (83% on rejected submissions and 50% on accepted ones, which were approximately one-quarter of the total submissions) and preliminary analysis suggests standard deviations of about 5% and 13% in the agreement on rejection and acceptance decisions respectively.^c These results are not earth-shattering—prior to the experiment Cortes and Lawrence predicted an agreement of 75% and 80% (respectively)—and so one interpretation is they simply confirm what many of us believe—that there is a significant subjective element to the peer review process. I see this as yet another reason to favor venues with rotating gatekeepers.

Are conferences perfect? Not by a long shot—for example, I have been involved in discussions^d on how to improve the experience for participants in one of the top theory conferences and I will be the first to admit that some of these issues do stem from the publication-venue role of the conferences. The reviewing process itself can be improved as well, and a lot of it depends on the diligence of the particular program chair and committee members.

The boundaries between conferences and journals are not that cut and dry. A number of communities have

I completely agree with many critics of our publication culture that we can and should be thinking of ways to improve it.

been exploring journal-conference “hybrid” models that can be of great interest. My sense is that conferences are better at *highlighting* the works that are of broad interest to the community (a.k.a. “reviewing” the paper), while journals do a better job at *verifying* the correctness and completeness of the paper (a.k.a. “refereeing”), and iterating with the author to develop more polished final results.

These are two different goals and are best achieved by different processes. For selecting particular works to highlight, comparing a batch of submissions by a panel of experts relying on many short reviews (as is the typical case in a conference) seems to work quite well. But fewer deeper reviews, involving a back-and-forth between author and reviewer (as is ideally the case in a journal) are better at producing a more polished work, and one in which we have more confidence in its correctness. We can try to find ways to achieve the best of both worlds, and make the most efficient use of the community’s attention span and resources for refereeing. I personally like the “integrated journal/conference” model where a journal automatically accepts papers that appeared in certain conferences, jumping straight into the revision stage, which can involve significant interaction with the author. The advantage is that by outsourcing the judgment of impact and interest to the conference, the journal review process avoids redundant work and can be focused on the roles of verifying correctness and improving presentation. Moreover, the latter properties are more objective, and hence the process can be somewhat less “adversarial” and involve more junior referees such as stu-

dents. In fact, in many cases these referees could dispense with anonymity and get some credit in print for their work.

Perhaps the biggest drawback of conferences is the cost in time and resources to attend them. This is even an issue for “top tier” conferences, where this effort at least pays off for attendees who get to hear talks on exciting new works as well as connect with many others in their community. But it is a greater problem for some lower-ranked conferences where many participants only come when they present a paper, and in such a case it may indeed have been better off if those papers appeared in a journal. In fact, I wish it were acceptable for researchers’ work to “count” even if it appeared in neither a conference nor a journal. Some papers can be extremely useful to experts working in a specific field, but have not yet advanced to a state where they are of interest to the broader community. We should think of ways to encourage people to post such works online without spending resources on refereeing or travel. While people often lament the rise of the “least publishable unit,” there is no inherent harm (and there is some benefit) in researchers posting the results of their work, no matter how minor they are. The only problem is the drain on resources when these incremental works go through the peer review process. Finally, open access is of course a crucial issue and I do believe^e both conferences and journals should make all papers, most of which represent work supported by government grants or non-profit institutions, freely available to the public.

To sum up, I completely agree with many critics of our publication culture that we can and should be thinking of ways to improve it. However, while doing so we should also acknowledge and preserve the many positive aspects of our culture, and take care to use the finite resource of quality refereeing in the most efficient manner. **□**

^e See the author’s December 2012 blog post: <http://bit.ly/1UcYdFF>

^c See the March 2015 blog post by Neil Lawrence: <http://bit.ly/1pK4Anr>

^d See the author’s May 2015 blog post: <http://bit.ly/1pK4LiF>

Boaz Barak (b@boazbarak.org) is the Gordon McKay Professor of Computer Science at the Harvard John A. Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, MA.

Viewpoint

Privacy Is Dead, Long Live Privacy

Protecting social norms as confidentiality wanes.

THE PAST FEW years have been especially turbulent for privacy advocates. On the one hand, the global dragnet of surveillance agencies has demonstrated the sweeping surveillance achievable by massively resourced government organizations. On the other, the European Union has issued a mandate that Google definitively “forget” information in order to protect users.

Privacy has deep historical roots, as illustrated by the pledge in the Hippocratic oath (5th century B.C.), “Whatever I see or hear in the lives of my patients ... which ought not to be spoken of outside, I will keep secret, as considering all such things to be private.”¹¹ Privacy also has a number of definitions. A now common one among scholars views it as the flow of information in accordance with social norms, as governed by context.¹⁰ An intricate set of such norms is enshrined in laws, policies, and ordinary conduct in almost every culture and social setting. Privacy in this sense includes two key notions: confidentiality and fair use. We argue that confidentiality, in the sense of individuals’ ability to preserve secrets from governments, corporations, and one another, could well continue to erode. We call instead for more attention and research devoted to fair use.

To preserve existing forms of privacy against an onslaught of online threats, the technical community is



working hard to develop *privacy-enhancing technologies* (PETs). PETs enable users to encrypt email, conceal their IP addresses, avoid tracking by Web servers, hide their geographic location when using mobile devices, use anonymous credentials, make untraceable database queries, and publish documents anonymously. Nearly all major PETs aim at protecting confidentiality; we call these *confidentiality-oriented PETs* (C-PETs). C-PETs

can be good and helpful. But there is a significant chance that in many or most places, C-PETs will not save privacy. It is time to consider adding a new research objective to the community’s portfolio: preparedness for a post-confidentiality world in which many of today’s social norms regarding the flow of information are regularly and systematically violated.

Global warming offers a useful analogy, as another slow and seem-

ingly unstoppable human-induced disaster and a worldwide tragedy of commons. Scientists and technologists are developing a portfolio of mitigating innovations in renewable energy, energy efficiency, and carbon sequestration. But they are also studying ways of coping with likely effects, including rising sea levels and displacement of populations. There is a scientific consensus that the threat justifies not just mitigation, but preparation (for example, elevating Holland's dikes).

The same, we believe, could be true of privacy. Confidentiality may be melting away, perhaps inexorably: soon, a few companies and surveillance agencies could have access to most of the personal data of the world's population. Data provides information, and information is power. An information asymmetry of this degree and global scale is an absolute historical novelty.

There is no reason, therefore, to think of privacy as we conceive of it today as an enduring feature of life.

Example: RFID

Radio-Frequency Identification (RFID) location privacy concretely illustrates how technological evolution can undermine C-PETs. RFID tags are wireless microchips that often emit static identifiers to nearby readers. Numbering in the billions, they in principle permit secret local tracking of ordinary people. Hundreds of papers proposed C-PETs that rotate identifiers to prevent RFID-based tracking.⁶

Today, this threat seems quaint. Mobile phones with multiple RF interfaces (including Bluetooth, Wi-Fi, NFC), improvements in face recognition, and a raft of new wireless devices (fitness trackers, smartwatches, and other devices), offer far more effective ways to track people than RFID ever did. They render RFID C-PETs obsolete.

This story of multiplying threat vectors undermining C-PETs' power—and privacy more generally—is becoming common.

The Assault on Privacy

We posit four major trends providing the means, motive, and opportunity for the assault on privacy in its broad-

est sense. The adversaries include surveillance agencies and companies in markets such as targeted advertising, as well as smaller, nefarious players.

Pervasive data collection. As the number of online services and always-on devices grows, potential adversaries can access a universe of personal data quickly expanding beyond browsing history to location, financial transactions, video and audio feeds, genetic data⁴, real-time physiological data—and perhaps eventually even brainwaves.⁸ These adversaries are developing better and better ways to correlate and extract new value from these data sources, especially as advances in applied machine learning make it possible to fill in gaps in users' data via inference. Sensitive data might be collected by a benevolent party for a purpose that is acceptable to a user, but later fall into dangerous hands, due to political pressure, a breach, and other reasons. "Secondhand" data leakage is also growing in prevalence, meaning that one person's action impacts another's private data (for example, if a friend declares a co-location with us, or if a blood relative unveils her genome). The emerging Internet of Things will make things even trickier, soon surrounding us with objects that can report on what we touch, eat, and do.¹⁶

Monetization (greed). Political philosophers are observing a drift from what they term *having* a market economy to *being* a market society¹³ in which market values eclipse non-market social norms. On the Internet, the ability to monetize nearly every piece of information is clearly fueling this process, which is itself facilitated by the existence of quasi-monopolies. A market-

There is no reason to think of privacy as we conceive of it today as an enduring feature of life.

place could someday arise that would seem both impossible and abhorrent today. (For example, for \$10: "I know that Alice and Bob met several times. Give me the locations and transcripts of their conversations.") Paradoxically, tools such as anonymous routing and anonymous cash could facilitate such a service by allowing operation from loosely regulated territories or from no fixed jurisdiction at all.

Adaptation and apathy. Users' data curation habits are a complex research topic, but there is a clear generational shift toward more information sharing, particularly on social networks. (Facebook has more than one billion users regularly sharing information in ways that would have been infeasible or unthinkable a generation ago.). Rather than fighting information sharing, users and norms have rapidly changed, and convenience has trumped privacy to create large pockets of data-sharing apathy. Foursquare and various other microblogging services that encourage disclosure of physical location, for example, have led many users to cooperate in their own physical tracking. Information overload has in any event degraded the abilities of users to curate their data, due to the complex and growing challenges of "secondhand" data-protection weakening and inference, as noted previously.

Secret judgment. Traceability and accountability are essential to protecting privacy. Facebook privacy settings are a good example of visible privacy practice: stark deviation from expected norms often prompts consumer and/or regulatory pushback.

Increasingly often, though, sensitive-data exploitation can happen away from vigilant eyes, as the recent surveillance scandals have revealed. (National security legitimately demands surveillance, but its scope and oversight are critical issues.) Decisions made by corporations—hiring, setting insurance premiums, computing credit ratings, and so forth—are becoming increasingly algorithmic, as we discuss later. Predictive consumer scores are one example; privacy scholars have argued they constitute a regime of secret, arbitrary, and potentially discriminatory and abusive judgment of consumers.²

A Post-Confidentiality Research Agenda

We should prepare for the possibility of a post-confidentiality world, one in which confidentiality has greatly eroded and in which data flows in such complicated ways that social norms are jeopardized. The main research challenge in such a world is to preserve social norms, as we now explain.

Privacy is important for many reasons. A key reason, however, often cited in discussions of medical privacy, is concern about abuse of leaked personal information. It is the potentially resulting unfairness of decision making, for example, hiring decisions made on the basis of medical history, that is particularly worrisome. A critical, defensible bastion of privacy we see in post-confidentiality world therefore is in the *fair use* of disclosed information.

Fair use is increasingly important as algorithms dictate the fates of workers and consumers. For example, for several years, some Silicon Valley companies have required job candidates to fill out questionnaires (“Have you ever set a regional-, state-, country-, or world-record?”). These companies apply classification algorithms to the answers to filter applications.⁵ This trend will surely continue, given the many domains in which statistical predictions demonstrably outperform human experts.⁷ Algorithms, though, enable deep, murky, and extensive use of information that can exacerbate the unfairness resulting from disclosure of private data.

On the other hand, there is hope that algorithmic decision making can lend itself nicely to protocols for enforcing accountability and fair use. If decision-making is algorithmic, it is possible to require decision-makers to prove that they are not making use of information in contravention of social norms expressed as laws, policies, or regulations. For example, an insurance company might prove it has set a premium without taking genetic data into account—even if this data is published online or otherwise widely available. If input data carries authenticated labels, then cryptographic techniques permit the construction of such proofs without revealing underlying algorithms, which may themselves be company

If we cannot win the privacy game definitively, we need to defend paths to an equitable society.

secrets (for example, see Ben-Sasson et al.¹). Use of information flow control¹² preferably enforced by software attested to by a hardware root of trust (for example, see McKeen et al.⁹) can accomplish much the same end. Statistical testing is an essential, complementary approach to verifying fair use, one that can help identify cases in which data labeling is inadequate, rendered ineffective by correlations among data, or disregarded in a system. (A variety of frameworks exist, for example, see Dwork et al.³)

Conclusion

A complementary research goal is related to privacy quantification. To substantiate claims about the decline of confidentiality, we must measure it. Direct, global measurements are difficult, but research might look to indirect monetary ones: The profits of the online advertising industry per pair of eyeballs and the “precision” of advertising, perhaps as measured by click-through rates. At the local scale, research is already quantifying privacy (loss) in such settings as location-based services.¹⁴

There remains a vital and enduring place for confidentiality. Particularly in certain niches—protecting political dissent, anti-censorship in repressive regimes—it can play a societally transformative role. It is the responsibility of policymakers and society as a whole to recognize and meet the threat of confidentiality’s loss, even as market forces propel it and political leaders give it little attention. But it is also incumbent upon the research community to contemplate alternatives to C-PETs, as confidentiality is broadly menaced by technology and

social evolution. If we cannot win the privacy game definitively, we need to defend paths to an equitable society. We believe the protection of social norms, especially through fair use of data, is the place to start. While C-PETs will keep being developed and will partially mitigate the erosion of confidentiality, we hope to see many “fair-use PETs” (F-PETs) proposed and deployed in the near future.¹⁵ **□**

References

1. Ben-Sasson, E. et al. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *Advances in Cryptology—CRYPTO*, (Springer, 2013), 90–108.
2. Dixon, P. and Gellman, R. The scoring of America: How secret consumer scores threaten your privacy and your future. Technical report, World Privacy Forum (Apr. 2, 2014).
3. Dwork, C. et al. Fairness through awareness. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. (ACM, 2012), 214–226.
4. Erlich, Y. and Narayanan, A. Routes for breaching and protecting genetic privacy. *Nature Reviews Genetics* 15, 6 (2014), 409–421.
5. Hansell, S. Google answer to filling jobs is an algorithm. *New York Times* (Jan. 3, 2007).
6. Juels, A. RFID security and privacy: A research survey. *IEEE Journal on Selected Areas in Communication* 24, 2 (Feb. 2006).
7. Kahneman, D. *Thinking, Fast and Slow*. Farrar, Straus, and Giroux, 2012, 223–224.
8. Martinovic, I. et al. On the feasibility of side channel attacks with brain-computer interfaces. In *Proceedings of the USENIX Security Symposium*, (2012), 143–158.
9. McKeen, F. et al. Innovative instructions and software model for isolated execution. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, Article no. 10 (2013).
10. Nissenbaum, H. *Privacy in Context: Technology, Policy, and the Integrity of Social Life*. Stanford University Press, 2009.
11. North, M.J. Hippocratic oath translation. U.S. National Library of Medicine, 2002.
12. Sabelfeld, A. and Myers, C. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications* 21, 1 (2003), 5–19.
13. Sandel, M.J. *What Money Can't Buy: The Moral Limits of Markets*. Macmillan, 2012.
14. Shokri, R. et al. Quantifying location privacy. In *Proceedings of the IEEE Symposium on Security and Privacy* (2011), 247–262.
15. Tramèr, F. et al. Discovering Unwarranted Associations in Data-Driven Applications with the FairTest Testing Toolkit, 2016; arXiv:1510.02377.
16. Weber, R.H. Internet of things—New security and privacy challenges. *Computer Law and Security Review* 26, 1 (2010), 23–30.

Jean-Pierre Hubaux (jean-pierre.hubaux@epfl.ch) is a professor in the Computer Communications and Applications Laboratory at the Ecole Polytechnique Fédérale de Lausanne in Switzerland.

Ari Juels (juels@cornell.edu) is a professor at Cornell Tech (Jacobs Institute) in New York.

We would like to thank George Danezis, Virgil Gligor, Kévin Huguenin, Markus Jakobsson, Huang Lin, Tom Ristenpart, Paul Syverson, Gene Tsudik and the reviewers of this Viewpoint for their many generously provided, helpful comments, as well as the many colleagues with whom we have shared discussions on the topic of privacy. The views presented in this Viewpoint remain solely our own.

Copyright held by authors.

Viewpoint

A Byte Is All We Need

A teenager explores ways to attract girls into the magical world of computer science.

IT WAS TIME to begin teaching my class. The children were in their seats, laptops turned on, ready to begin. I scanned the doorway, hoping for one more girl to arrive: there were nine boys in my class and just two girls. I was conducting *free* coding classes, but young girls were still reluctant to attend. As a 15-year-old computer enthusiast, I was baffled by this lack of interest.

A young boy arrived with his mother. As the mother was preparing to leave, I asked her, “If you have the time, why don’t you stay? Maybe you could help your son.” She agreed. I started my class without further delay. In the next class, the boy’s mother brought along a friend and her daughter. Subsequent classes saw the registration of a few more girls, friends of friends. My message was getting across: computer science (CS) is not as difficult as presumed—it is fun, and more importantly, it is certainly not an exclusively male-oriented domain.

Gender Difference in Perspectives

Being enamored by CS myself, I was disappointed to find girls shunned this super-exciting, super-useful, and super-pervasive discipline. I was determined to find out why, and as I started teaching Java to middle school children I kept a close watch on how the questions, understanding, reactions, and study methods of girls differed from the boys in class. The difference I noticed immediately was the boys were more advanced in their knowledge. It was a challenge for me to balance the boys and the girls not only



Exposure and encouragement are key to attracting girls to CS: the author doing her part.

in teaching but also in their learning perspectives. I noted that while the boys accepted concepts unquestioningly and focused on application—the ‘How’ of things—the girls always wanted to know ‘Why?’ So I asked the boys to explain the ‘why’ of things to the girls. The boys soon learned they did not know it all, so attempted a deeper understanding and in the process the girls got their answers. By the time the session was over, both boys and girls were equivalent in knowledge and confidence, and were keen to collaborate in writing apps.

Dive In Early

But why was there so much disparity at the start? After a brief round of questioning, I realized the boys had a head start because they had started young—just like I had. Young boys are more attracted to computer games and gadgets than young girls. As I have an older brother, I had been exposed to

computer games and programming as a small child. But what about girls with no brothers? Girls are not aware of the fun element in controlling computers most often because they have not had the opportunity to try it.

The essential difference between the genders in the interest and knowledge in computer science stems from exposure (or the lack thereof) at a young age. If one goes to a store to buy PlayStation, Nintendo, or Xbox games, the gender imbalance is apparent. Except for a few Barbie games, there are practically no games with young girls as protagonists. There have been a few attempts to create stimulating games geared only for girls. In 1995, Brenda Laurel started her company Purple Moon to make video games that focused particularly on girls’ areas of interest while retaining the action and challenge mode. Despite extensive research on the interests and inclinations of girls, Purple Moon failed.⁴ To-

day, the Internet has games for young girls but most are based on cultural biases like dressing up, cooking, nail art, fashion designing, and shopping. Exciting and challenging video games continue to be male oriented, which makes the initiation into computer science easier and earlier for boys. Once hooked on these games, curiosity and the wish to engineer desired results take the boys into the world of programming. And that is the bit that starts the coder's journey.

It is a journey whose momentum can be picked up by girls, too. Facebook COO Sheryl Sandberg says, "Encourage your daughters to play video games." She claims, "A lot of kids code because they play games. Give your daughters computer games." In a gaming world thirsting for young girls' games, there are some invigorating splashes, like the wonderfully created game *Child of Light*.⁵ This role-playing game not only has a little girl as the central character but most of its other characters (both good and bad) are women, too. It is this kind of game the world needs to entice girls into the world of computer science—a world where women can have fun, create, and lead. The lead programmer of *Child of Light*, Brie Code says, "It can be lonely to be the only woman or one of very few women on the team ... it is worth pushing for more diversity within the industry."

Play to Learn— Replace Fear with Fun

Computer games for the very young have a vital role to play in ushering in diversity within the industry. Fred Rogers, the American icon for children's entertainment and education, rightly said, "For children, play is serious learning." I learned the program LOGO when I was just four years old because it was only a game to me—I was not programming, I was having fun. To be able to control the movements of the LOGO turtle was thrilling. Today, when I code in Java to make complex apps I gratefully acknowledge the little LOGO turtle that started it all. That was 10 years ago. Today, more exciting programming languages like *KIBO*, *Tynker*, and *ScratchJr*, software like *The Foos* and apps like *Kodable* aim to make computer science fun for small chil-



Early interest in computer science gives boys a head start.

dren who have not yet learned to read! This is the level at which girls have to enter the field of CS, not hover around the boundaries at high school. In the 21st century, computer science is as much a part of fundamental literacy as reading, writing, and math.

Not an Option, Mandatory K–12 Learning

Hence, I believe CS should be made mandatory in kindergarten and elementary school. President Obama has stressed the need for K–12 computer science education to "make them job-ready on day one." Learning the basics determines choices for higher studies. Girls need a bite and taste in early childhood in order to make informed decisions about computer science when they are teenagers. I was not a teenager yet when I migrated from India to the U.S. My private school in

India had CS as a compulsory subject from Grade 1 onward. So when I began attending school in the U.S., I knew I loved CS and it definitely had to be one of my electives. But the boys looked at us few girls in class as if we were aliens! Even today, in my AP computer science class, few boys ask me for a solution if they have a problem (I kid myself that it is because of my age and not gender). In India, it is cool for a girl to study computer science in school. I basked in virtual glory there, while in the U.S. I found most of my female friends raised their eyebrows, their eyes asking me, "Why would you want to study CS?"

So I decided to flip the question and conduct a survey to find out why they did *not* want to study computer science. I interviewed 107 girls from the ages of 5 to 17 in the U.S., U.K., and India. My question was: "Would you study computer science in college?" A whopping 82.4% of the girls said 'No' or 'Maybe'. When asked why not, 78% of them answered 'I am afraid that I am not smart enough to do CS.' Other answers included 'I am not a big fan of programming'/'I am not inclined toward the sciences, I am more creatively oriented'/'I prefer the literary field, writing, editing, publishing'/'I am too cool to be a geek'! When I asked whether they knew any programming language, only 14 girls out of the 107 said 'Yes.' Dismayed by the results, I posed the same question to 50 boys, in the same age group: 74.8% of them said 'Yes' to studying CS; 82% of all the boys I interviewed knew more than one programming language and many of them were less than 10 years old. My resolve



Starting young removes fear and makes coding fun.

was strengthened: the only way to remove the fear of CS from the minds of girls is to catch them young and encourage curiosity before negative attitudes are developed.

Thorns of Preconceived Notions

Once the worth of the field is realized, one notices the crop of roses and the thorns pale in comparison. One such thorn is the idea of geekiness that mars the face of computer science. Girls are unwilling to be nerds. The misconception that a computer nerd is a socially awkward eccentric has been obliterated by the stars in technology like Sheryl Sandberg, Sophie Wilson, Marissa Mayer, or the cool young female employees of Facebook, Google, and other companies and organizations.

Another (and a more piercing) thorn that keeps girls away from computer science is the lack of confidence in math and science. Intensive studies indicate spatial reasoning skills are deeply associated with mathematical and technical skills. Research also shows action and building games vastly improve spatial skills, math skills, divergent problem-solving skills, and even creativity. So why should these games be reserved only for boys? To develop spatial skills and attract girls into the tech field, Stanford engineers Bettina Chen and Alice Brooks created *Roominate*, a wired DIY dollhouse kit, where girls can build a house with building blocks and circuits, design and create rooms with walls, furniture, working fans, and lights.⁶ These kinds of toys can develop spatial perception and engender confidence in STEM fields in girls, too. “Playing an action video game can virtually eliminate gender difference in spatial attention and simultaneously decrease the gender disparity in mental rotation ability, a higher-level process in spatial cognition.”² The same study concludes, “After only 10 hours of training with an action video game, subjects realized substantial gains in both spatial attention and mental rotation, with women benefiting more than men. Control subjects who played a non-action game showed no improvement. Given that superior spatial skills are important in the mathematical and engineering sciences, these findings have practical implications for attracting men and women to these fields.”

It is just a matter of time before girls realize studying and working in CS is neither fearsome, nor boring.

Programming before Reading

Inspired by studies like these, I started a project to make CS appealing to the feminine mind. Based on my ‘Catch Them Young’ philosophy, I am using my programming knowledge to create action and building games for tiny tots where the action is determined by the player. The player decides whether the little girl protagonist wants to build a castle or rescue a pup from evil wolves. There is not a single alphabet used in the games, so that even two-year-old children can play with ease and develop their spatial reasoning even before they learn the alphabet. Using LOGO, ScratchJr, and Alice, I have created a syllabus to enable an early understanding of logic and creation of a sequence of instructions, which is the basis of all programming. I am currently promoting this course in private elementary schools in the U.S., India, and other countries to expose the minds of young girls to computer science.

Computational Thinking

In a similar endeavor, schools in South Fayette Township (near Pittsburgh, PA) have introduced coding, robotics, computer-aided design, 3D printing, and more, as part of the *regular curriculum* from kindergarten to 12th grade to make computational reasoning an integral part of thinking.¹ The problem-solving approach learned in the collaborative projects helps children apply computational thinking to the arts, humanities, and even English writing. Girls in these South Fayette schools are now computer whiz kids fixing computer bugs as well as malfunctioning hardware! They are clear proof that computational thinking is a strength not restricted to males alone—girls often combine it with creativity, designing, and literary skills for even more powerful effects. Recent stud-

ies show girls often go the extra length in creating more complex programs than boys. As Judith Good of the school of engineering and informatics at the University of Sussex commented after a workshop for boys and girls for creating computer games: “In our study, we found more girls created more scripts that were both more varied in terms of the range of actions they used, and more complex in terms of the computational constructs they contained.”³

Conclusion

The change has begun and it is just a matter of time before girls realize studying and working in CS is neither fearsome, nor boring. The time to realize this truth can be further shortened if parents and teachers can also be inducted into the realm of CS. Adult computer education is essential for closing the gender gap in computer science. The role of parents and teachers is paramount in introducing curiosity and interest in young children, irrespective of gender, and steer them toward the magic of CS. The huge participation of Indian mothers in the CS stream in Bangalore is a powerful stimulus and one of the primary reasons behind young girls embracing this field so enthusiastically and successfully in India. When role models open up new vistas of the computer science world to all children—including the very young—only then can the unfounded fear of girls regarding this relatively new domain be replaced by curiosity, excitement, and a desire to participate. For computational thinking has no gender—it just has magical power. 

References

1. Berdik, C. Can coding make the classroom better? *Slate* 23 (Nov. 23, 2015); <http://slate.me/1Sfwlwc>.
2. Feng, J., Spence, I., and Pratt, J. Playing an action video game reduces gender differences in spatial cognition. *Psychological Science* 18, 10 (Oct. 2007), 850–855; <http://bit.ly/1pmG8Am>.
3. Gray, R. Move over boys: Girls are better at creating computer games than their male friends. *DailyMail.com* (Dec. 2, 2014); <http://dailymail.ai/1FMtEN1>.
4. Harmon, A. With the best research and intentions, a game maker fails. *The New York Times* (Mar. 22, 1999); <http://nyti.ms/1V1hxEL>.
5. Kaszor, D. PS4 preview: Child of Light a personal project born within a giant game developer. *Financial Post* (Nov. 12, 2013).
6. Sugar, R. How failing a freshman year physics quiz helped 2 friends start a “Shark Tank” funded company. *Business Insider* 21 (Jul. 21, 2015); <http://read.bi/1SkHep>.

Ankita Mitra (ankita.m888@gmail.com) is a 15-year-old student at Monta Vista High School in Cupertino, CA.

Copyright held by author.



Article development led by **acmqueue**
queue.acm.org

Many of the skills aren't technical at all.

BY KATE MATSUDAIRA

Nine Things I Didn't Know I Would Learn Being an Engineer Manager

WHEN I MOVED from being an engineer to being a dev lead, I knew I had a lot to learn. My initial thinking was I had to be able to do thorough code reviews, design, and architect websites, see problems before they happened, and ask insightful technical questions. To me that meant learning the technology and

becoming a better engineer. When I actually got into the role (and after doing it almost 15 years), the things I have learned—and that have mattered the most—were not those technical details. In fact, many of the skills I have built that made me a good engineer manager were not technical at all and, while unexpected lessons, have helped me in many other areas of my life.

What follows are some of these les-

sons, along with ideas for applying them in your life—whether you are a manager, want to be a manager, or just want to be a better person and employee.

1. Driving consensus. Technical people love to disagree. I've found there usually are no definitive answers to a problem. Instead, there are different paths with different risks, and each solution has its own pros and cons. Being able to get people to agree (without being the dictator telling people what

to do) means learning how to get everyone on the same page.

Since meetings with lots of people can be very contentious, one technique that has helped me wrangle those ideas is called *multivoting*. Multivoting is helpful for narrowing a wide range of ideas down to a few of the most important or appropriate, and it allows every idea to receive consideration.

You can do this by first brainstorming as a team while putting all the ideas on a whiteboard, along with the pros and cons of each. From there you go through a voting process until the group arrives at what it considers to be an appropriate number of ideas for further analysis. Organizational development consultant Ava S. Butler explains the multivoting process in wonderful detail if you would like more information.¹

2. Bringing out ideas (even from quiet people). One of the challenges of working with introverts and shy people without strong communication skills is it can be difficult to surface their ideas. They tend to be quiet in meetings and keep their ideas (which can be very good!) to themselves. Here are a few techniques I have learned that help me bring these people out of their shells:

- In meetings I call on people or do a round robin so everyone gets a chance to talk. This way, the shy team members are given the floor to speak where they may have otherwise remained silent.

- In one-on-ones I have learned to use the power of silence. I ask a question and then refrain from speaking until the person answers—even if it is a minute later. I had to learn to get comfortable with uncomfortable silence, which has been a powerful technique in uncovering what people are thinking.

- I often have everyone write their ideas on a Post-it note and put it on the whiteboard during team meetings. This allows everyone's ideas to receive equal weight, and introverted people are therefore encouraged to share their thoughts.

3. Explaining tech to nontech. When you want to rewrite code that already works, you have to justify the change to management. Much of the time nontechnical people do not care about the details. Their focus is on results. Therefore, I have learned to look at all



my work, and the work my team does, in a business context. For example, does it save time, money, or generate revenue—and then how do I best communicate that?

I frame my ideas in a context that matters to the specific audience I am addressing. Using analogy is one technique I have found to be quite powerful.² Explaining an idea through analogy allows you to consider your audience's perspective and talk at their level, never above them.

4. Being a good listener. When you manage people you really must learn to listen. And, by the way, listening goes way beyond paying attention to what is said. You should also be paying attention to body language and behavior.

I like to use the example of an employee who always arrives early to work. If that person suddenly makes a new habit of showing up late, this could be a cue that something is amiss. By listening to that person's actions, and not just their words, you gain valuable insight and can manage with greater empathy and awareness.

5. Caring about appearance. When you are in a leadership role you often meet with people outside of your immediate co-workers who do not know you as well. And they judge you. Plus, studies have shown that your appearance strongly influences other people's perception of your intelligence, au-

thority, trustworthiness, financial success, and whether you should be hired or promoted.⁵

Growing up, I was taught by my grandfather how to dress for the job I wanted, not the job I currently had. As a new manager, I put more of an effort into my appearance, and it definitely had a positive effect, especially when interacting with customers and clients outside of the organization.

I recommend emulating the people in your organization whom you look up to. Look at how they dress. Study how they carry themselves. Watch how they conduct themselves in meetings, parties, and other events. This is where you can get your best ideas for how to dress and communicate success. You want your work and reputation to speak for itself, but do not let your appearance get in the way of that.

6. Caring about other disciplines. The more you know about other facets of the business, like sales and marketing, the more capable you are of making strategic decisions. The higher up you go, the more important this is, because you are not just running software—you are running a business.

It is also vital to understand the needs of your customers. You could build what you believe is an amazing product, but it could end up being useless to the customer if you never took the time to fully understand their



needs. Even if you work in back-end development, caring about the end user will make you create better solutions.

7. Being the best technologist does not make you a good leader. If you are managing enough people or products, you do not have time to dive into the deep details of the technology. Moreover, you need to learn to trust the people on your team. It is better to let them be the experts and shine in meetings than to spend your time looking over their shoulders to know all the details.

The best skills you can have are these:

- Ask great questions that get to the root of the problem. This helps others think through their challenges, uncovering issues before they arise.

- Delegate and defer so that you are able to accomplish more while empowering those around you.

- Teach people to think for themselves. Instead of prescribing answers, ask people what they think you would say or tell them to do. I highly recommend David Marquet's talk, "Greatness."³ He reveals that while working as a captain on a military submarine he vowed never to give another order. Instead, he allowed his reports to make their own empowered decisions. This small shift in thinking brought about powerful change.

8. Being organized and having a

system. When you are responsible for the work of others, you must have checks and balances. Practicing strong project-management skills is key. You must have a way of keeping things organized and know what is going on, and be able to communicate it when things are not going as planned.

It is also important to be strategic about your own time management. I start each week with at least 30 minutes dedicated to looking at my top priorities for the week, and then I carve out the time to make progress on these priorities. One time-management tool that has been successful for me is time blocking, where I plan my days in a way that optimizes my time for my productivity (for example, I am a much better writer in the mornings so I make sure to do my writing then).⁴ This helps me optimize my time and always know the best way to use a spare 15 minutes.

Similarly, I have a system for keeping track of my great ideas. I keep an Evernote where I save articles I love or interesting ideas I come across. This gives me a little vault of information I can go to when I need to get inspired, write a blog post, or come up with something worthwhile to post on social media.

The point here is to have systems in place. You need a way to do all the things that are important and keep your

information and details organized.

9. Networking. If you think about it, every job offer, promotion, and raise was not given to you because of the work you did. The quality of your work *may* have been a factor, but there was a person behind those decisions. It was *someone* who gave you those opportunities.

If you do great work and no one likes you, then you simply will not be as successful. Be someone with whom people want to work. For example, helping others, listening intently, and caring about the lives of the people around you will help you profoundly. I am always looking for ways to expand my network, while also deepening the relationships I have with my mentors and friends.

I hope these ideas help you become a better leader or employee. Pick one or two to focus on each week, and see where it takes you—progress is a process! I would love to hear from you, especially if you have any other ideas to add to this list. G

Related articles on queue.acm.org

Mal Managerium: A Field Guide

Phillip Laplante

<http://queue.acm.org/detail.cfm?id=1066076>

Sink or Swim, Know When It's Time to Bail

Gordon Bell

<http://queue.acm.org/detail.cfm?id=966806>

Adopting DevOps Practices in Quality Assurance

James Roche

<http://queue.acm.org/detail.cfm?id=2540984>

References

- Butler, A.S. Ten techniques to make decisions: #2 multivoting, 2014; <http://www.avasbutler.com/ten-techniques-to-make-decisions-2-multivoting/#.Vtd1ZYwrIy4>.
- Gavetti, G., Rivkin, J.W. How strategists really think: tapping the power of analogy. *Harvard Business Review* (April, 2005); <https://hbr.org/2005/04/how-strategists-really-think-tapping-the-power-of-analogy>.
- Marquet, D. Inno-iversity presents: greatness. YouTube, 2013; https://www.youtube.com/watch?v=OqmdLcyES_Q.
- Matsudaira, K. Seven proven ways to get more done in less time, 2015; <http://katemats.com/7-proven-ways-to-get-more-done-in-less-time/>.
- Smith, J. Here's how clothing affects your success. *Business Insider* (Aug. 19, 2014); <http://www.businessinsider.com/how-your-clothing-affects-your-success-2014-8>.

Kate Matsudaira (katemats.com) is the founder of her own company, Popforms. Previously she worked in engineering leadership roles at companies like Decide (acquired by eBay), Moz, Microsoft, and Amazon.

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



Article development led by [acmqueue](https://queue.acm.org)
queue.acm.org

This visualization of software execution is a new necessity for performance profiling and debugging.

BY BRENDAN GREGG

The Flame Graph

AN EVERYDAY PROBLEM in our industry is understanding how software is consuming resources, particularly CPUs. What exactly is consuming how much, and how did this change since the last software version? These questions can be answered using software profilers—tools that help direct developers to optimize their code and operators to tune their environment. The output of profilers can be verbose, however, making it laborious to study and comprehend. The flame graph provides a new visualization for profiler output and can make for much faster comprehension, reducing the time for root cause analysis.

In environments where software changes rapidly, such as the Netflix cloud microservice architecture, it is especially important to understand profiles quickly. Faster comprehension can also make the study of foreign software more successful, where one's skills, appetite, and time are strictly limited.

Flame graphs can be generated from the output of many different software profilers, including profiles for different resources and event types. Starting with CPU profiling, this article describes how flame graphs work, then looks at the real-world problem that led to their creation.





CPU Profiling

A common technique for CPU profiling is the sampling of stack traces, which can be performed using profilers such as Linux `perf_events` and DTrace. The stack trace is a list of function calls that show the code-path ancestry. For example, the following stack trace shows each function as a line, and the top-down ordering is child to parent:

```
SpinPause
StealTask::do_it
GCTaskThread::run
java_start
start_thread
```

Balancing considerations that include sampling overhead, profile size, and application variation, a typical CPU profile might be collected in the following way: stack traces are sampled at a rate of 99 times per second (not 100, to avoid lock-step sampling) for 30 seconds across all CPUs. For a 16-CPU system, the resulting profile would contain 47,520 stack-trace samples. As text, this would be hundreds of thousands of lines.

Fortunately, profilers have ways to condense their output. DTrace, for example, can measure and print unique stack traces, along with their occurrence count. This approach is more

effective than it might sound: identical stack traces may be repeated during loops or when CPUs are in the idle state. These are condensed into a single stack trace with a count.

Linux `perf_events` can condense profiler output even further: not only identical stack trace samples, but also subsets of stack traces can be coalesced. This is presented as a tree view with counts or percentages for each code-path branch, as shown in Figure 1.

In practice, the output summary from either DTrace or `perf_events` is sufficient to solve the problem in many cases, but there are also cases where

the output produces a wall of text, making it difficult or impractical to comprehend much of the profile.

The Problem

The problem that led to the creation of flame graphs was application performance on the Joyent public cloud.³ The application was a MySQL database that was consuming around 40% more CPU resources than expected.

DTrace was used to sample user-

mode stack traces for the application at 997Hz for 60 seconds. Even though DTrace printed only unique stack traces, the output was 591,622 lines long, including 27,053 unique stack traces. Fortunately, the last screenful—which included the most frequently sampled stack traces—looked promising, as shown in Figure 2.

The most frequent stack trace included a MySQL `calc_sum_of_all_status()` function, indicating it was

processing a “show status” command. Perhaps the customer had enabled aggressive monitoring, explaining the higher CPU usage?

To quantify this theory, the stack-trace count (5,530) was divided into the total samples in the captured profile (348,427), showing it was responsible for only 1.6% of the CPU time. This alone could not explain the higher CPU usage. It was necessary to understand more of the profile.

Browsing more stack traces became an exercise in diminishing returns, as they progressed in order from most to least frequent. The scale of the problem is evident in Figure 3, where the entire DTrace output becomes a featureless gray square.

With so much output to study, solving this problem within a reasonable time frame began to feel insurmountable. There had to be a better way.

I created a prototype of a visualization that leveraged the hierarchical nature of stack traces to combine common paths. The result is shown in Figure 4, which visualizes the same output as in Figure 3. Since the visualization explained why the CPUs were “hot” (busy), I thought it appropriate to choose a warm palette. With the warm colors and flame-like shapes, these visualizations became known as flame graphs. (An interactive version of Figure 4, in SVG [scalable vector graphics] format is available at <http://queue.acm.org/downloads/2016/Gregg4.svg>.)

The flame graph allowed the bulk of the profile to be understood very quickly. It showed the earlier lead, the MySQL status command, was responsible for only 3.28% of the profile when all stacks were combined. The bulk of the CPU time was consumed in MySQL join, which provided a clue to the real problem. The problem was located and fixed, and CPU usage was reduced by 40%.

Flame Graphs Explained

A flame graph visualizes a collection of stack traces (aka call stacks), shown as an adjacency diagram with an inverted icicle layout.⁷ Flame graphs are commonly used to visualize CPU profiler output, where stack traces are collected using sampling.

A flame graph has the following characteristics:

Figure 1. Sample Linux `perf_events` tree view.

```
# perf report -n --stdio
[...]
```

#	Overhead	Samples	Command	Shared Object	Symbol
#	16.90%	490	dd	[kernel.kallsyms]	[k] xen_hypervisor_xen_version
					--- xen_hypervisor_xen_version
					check_events
					---97.76%-- extract_buf
					extract_entropy_user
					urandom_read
					vfs_read
					sys_read
					system_call_fastpath
					_GI__libc_read
					---0.82%-- _GI__libc_write
					---0.82%-- _GI__libc_read
					---0.61%-- [...]
	5.83%	169	dd	[kernel.kallsyms]	[k] sha_transform
					--- sha_transform
					extract_buf
					extract_entropy_user
					urandom_read
					vfs_read
					sys_read
					system_call_fastpath
					_GI__libc_read
					[...]

Figure 2. MySQL DTrace profile subset.

```
# dtrace -x ustackframes=100 -n 'profile-997 /execname == "mysqld"/ {
    @[ustack()] = count(); } tick-60s { exit(0); }'
dtrace: description 'profile-997' matched 2 probes
CPU    ID                FUNCTION:NAME
1      75195              :tick-60s
[...]
```

```
lib.so.1`__prctlset+0xa
lib.so.1`getparam+0x83
lib.so.1`pthread_getschedparam+0x3c
lib.so.1`pthread_setschedprio+0x1f
mysqld`_Z16dispatch_command19enum_server_commandP3THDPcj+0x9ab
mysqld`_Z10do_commandP3THD+0x198
mysqld`handle_one_connection+0x1a6
lib.so.1`_thrp_setup+0x8d
lib.so.1`_lwp_start
4884

mysqld`_Z13add_to_statusP17system_status_varS0_+0x47
mysqld`_Z22calc_sum_of_all_statusP17system_status_var+0x67
mysqld`_Z16dispatch_command19enum_server_commandP3THDPcj+0x1222
mysqld`_Z10do_commandP3THD+0x198
mysqld`handle_one_connection+0x1a6
lib.so.1`_thrp_setup+0x8d
lib.so.1`_lwp_start
5530
```

► A stack trace is represented as a column of boxes, where each box represents a function (a stack frame).

► The y-axis shows the stack depth, ordered from root at the bottom to leaf at the top. The top box shows the function that was on-CPU when the stack trace was collected, and everything beneath that is its ancestry. The function beneath a function is its parent.

► The x-axis spans the stack trace collection. It does *not* show the passage of time, so the left-to-right ordering has no special meaning. The left-to-right ordering of stack traces is performed alphabetically on the function names, from the root to the leaf of each stack. This maximizes box merging: when identical function boxes are horizontally adjacent, they are merged.

► The width of each function box shows the frequency at which that function was present in the stack traces, or part of a stack trace ancestry. Functions with wide boxes were more present in the stack traces than those with narrow boxes, in proportion to their widths.

► If the width of the box is sufficient,

it displays the full function name. If not, either a truncated function name with an ellipsis is shown, or nothing.

► The background color for each box is not significant and is picked at random to be a warm hue. This randomness helps the eye differentiate boxes, especially for adjacent thin

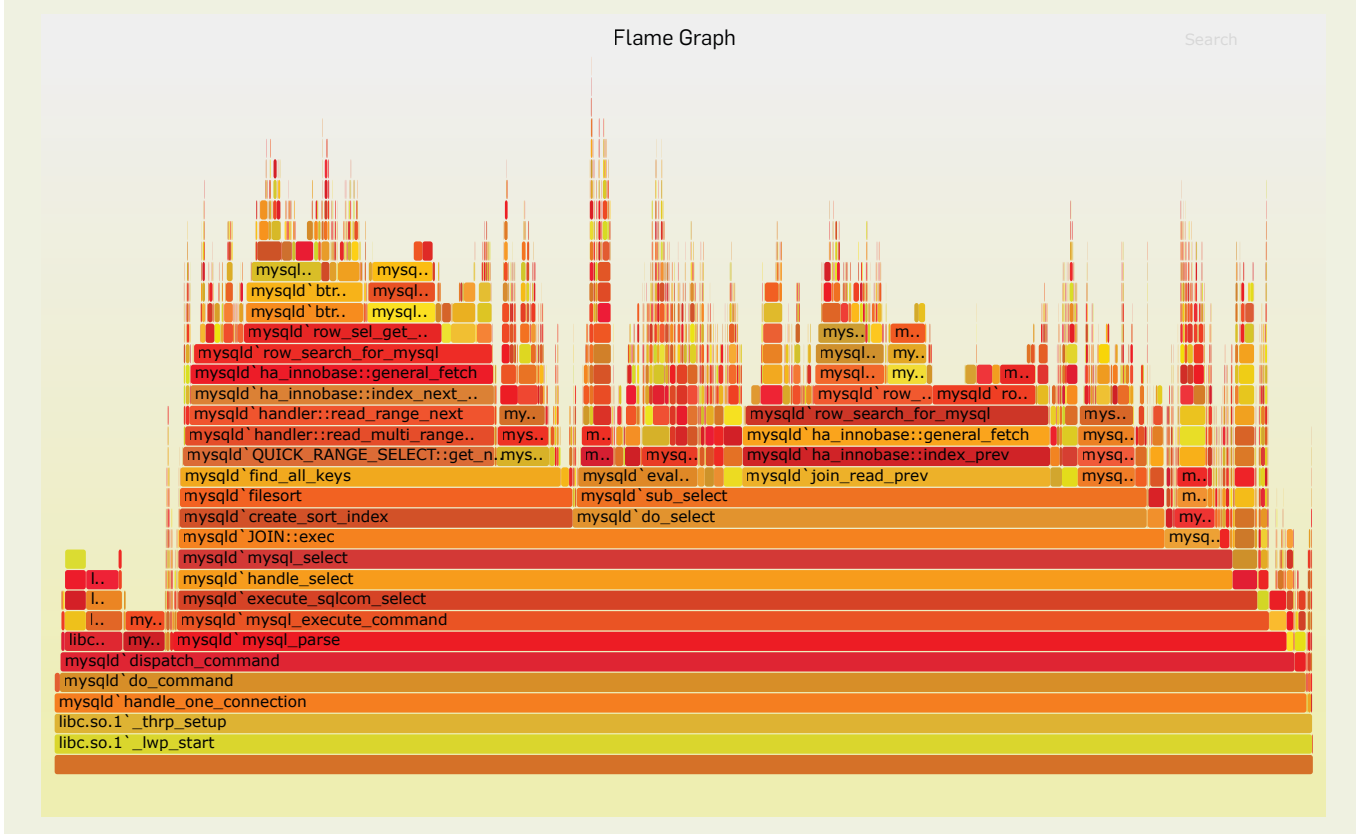
“towers.” Other color schemes are discussed later.

► The profile visualized may span a single thread, multiple threads, multiple applications, or multiple hosts. Separate flame graphs can be generated if desired, especially for studying individual threads.

Figure 3. Full MySQL DTrace profile output.



Figure 4. Full MySQL profiler output as a flame graph.



► Stack traces may be collected from different profiler targets, and widths can reflect measures other than sample counts. For example, a profiler (or tracer) could measure the time a thread was blocked, along with its stack trace. This can be visualized as a flame graph, where the *x*-axis spans the total blocked time, and the flame graph shows the blocking code paths.

As the entire profiler output is visualized at once, the end user can navigate intuitively to areas of interest. The shapes and locations in the flame graphs become visual maps for the execution of software.

While flame graphs use interactivity to provide additional features, these characteristics are fulfilled by a static flame graph, which can be shared as an image (for example, a PNG file or printed on paper). While only wide boxes have enough room to contain the function label text, they are also sufficient to show the bulk of the profile.

Interactivity

Flame graphs can support interactive features to reveal more detail, improve navigation, and perform calculations.

The original implementation of flame graphs⁴ creates an SVG image with embedded JavaScript for interactivity, which is then loaded in a browser. It supports three interactive features: Mouse-over for information, click to zoom, and search.

Mouse-over for information. On mouse-over of boxes, an informational line below the flame graph and a tooltip display the full function name, the number of samples present in the profile, and the corresponding percentage for those samples in the profile. For example, `Function: mysqld`JOIN::exec (272,959 samples, 78.34%)`.

This is useful for revealing the function name from unlabeled boxes. The percentage also quantifies code paths in the profile, which helps the user prioritize leads and estimate improvements from proposed changes.

Click to zoom. When a box is clicked, the flame graph zooms horizontally. This reveals more detail, and often function names for the child functions. Ancestor frames below the clicked box are shown with a faded background as a visual clue that their

Flame graphs can support interactive features to reveal more detail, improve navigation, and perform calculations.

widths are now only partially shown. A Reset Zoom button is included to return to the original full profile view. Clicking any box while zoomed will reset the zoom to focus on that new box.

Search. A search button or keystroke (Ctrl-F) prompts the user for a search term, which can include regular expressions. All function names in the profile are searched, and any matched boxes are highlighted with magenta backgrounds. The sum of matched stack traces is also shown on the flame graph as a percentage of the total profile, as in Figure 5. (An interactive version of Figure 5 in SVG format is available at <http://queue.acm.org/downloads/2016/Gregg5.svg>.)

This is useful not just for locating functions, but also for highlighting logical groups of functions—for example, searching for “`^ext4_`” to find the Linux `ext4` functions.

For some flame graphs, many different code paths may end with a function of interest—for example, spinlock functions. If this appeared in 20 or more locations, calculating their combined contribution to the profile would be a tedious task, involving finding then adding each percentage. The search function makes this trivial, as a combined percentage is calculated and shown on screen.

Instructions

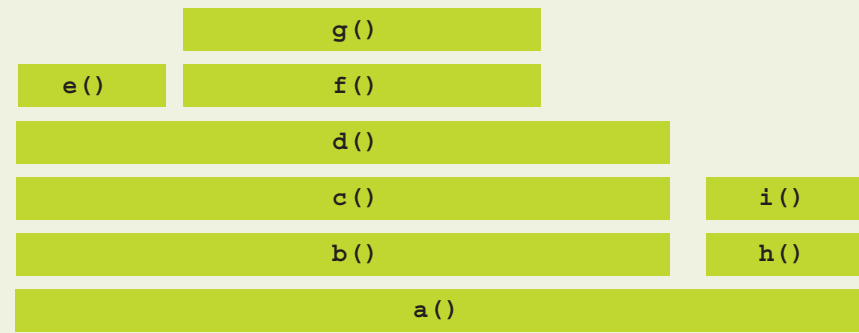
There are several implementations of flame graphs so far.⁵ The original implementation, `FlameGraph`,⁴ was written in the Perl programming language and released as open source. It makes the generation of flame graphs a three-step sequence, including the use of a profiler:

1. Use a profiler to gather stack traces (for example, Linux `perf_events`, `DTrace`, `Xperf`).
2. Convert the profiler output into the “folded” intermediate format. Various programs are included with the `FlameGraph` software to handle different profilers; their program names begin with “`stackcollapse.`”

3. Generate the flame graph using `flamegraph.pl`. This reads the previous folded format and converts it to an SVG flame graph with embedded JavaScript.

The folded stack-trace format puts stack traces on a single line, with functions separated by semicolons, followed by a space and then a count. The name

Figure 6. Example for interpretation.



below it, and so on. A quick scan downward from a function identifies why it was called.

- ▶ Reading bottom up shows code flow and the bigger picture. A function calls any child functions shown above it, which, in turn, call functions shown above them. Reading bottom up also shows the big picture of code flow before various forks split execution into smaller towers.

- ▶ The width of function boxes can be directly compared: wider boxes mean a greater presence in the profile and are the most important to understand first.

- ▶ For CPU profiles that employ timed sampling of stack traces, if a function box is wider than another, this may be because it consumes more CPU per function call or that the function was simply called more often. The function-call count is not shown or known via sampling.

- ▶ Major forks in the flame graph, spotted as two or more large towers atop a single function, can be useful to study. They can indicate a logical grouping of code, where a function processes work in stages, each with its own function. It can also be caused by a conditional statement, which chooses which function to call.

Interpretation Example

As an example of interpreting a flame graph, consider the mock one shown in Figure 6. Imagine this is visualizing a CPU profile, collected using timed samples of stack traces (as is typical).

The top edge shows that function `g()` is on-CPU the most; `d()` is wider, but its exposed top edge is on-CPU the least. Functions including `b()` and `c()` do not appear to have been sampled

on-CPU directly; rather, their child functions were running.

Functions beneath `g()` show its ancestry: `g()` was called by `f()`, which was called by `d()`, and so on.

Visually comparing the widths of functions `b()` and `h()` shows the `b()` code path was on-CPU about four times more than `h()`. The actual functions on-CPU in each case were their children.

A major fork in the code paths is visible where `a()` calls `b()` and `h()`. Understanding why the code does this may be a major clue to its logical organization. This may be the result of a conditional (if conditional, call `b()`, else call `h()`) or a logical grouping of stages (where `a()` is processed in two parts: `b()` and `h()`).

Other Code-Path Visualizations

As was shown in Figure 1, Linux `perf_events` prints a tree of code paths with percentage annotations. This is another type of hierarchy visualization: An indented tree layout.⁷ Depending on the profile, this can sometimes sufficiently summarize the output but not always. Unlike flame graphs, one cannot zoom out to see the entire profile and still make sense of this text-based visualization, especially after the percentages can no longer be read.

KCacheGrind¹⁴ visualizes code paths from profile data using a directed acyclic graph. This involves representing functions as labeled boxes (where the width is scaled to fit the function name), parent-to-child relationships as arrows, and then profile data is annotated on the boxes and arrows as percentages with bar chart-like icons. Similar to the problem with `perf_events`, if the visualization is zoomed out to fit a complex profile, then the annotations may no longer be legible.

The sunburst layout is equivalent to the icicle layout as used by flame graphs, but it uses polar coordinates.⁷ While this can generate interesting shapes, there are some difficulties: function names are more difficult to draw and read from sunburst slices than they are in the rectangular flame-graph boxes. Also, comparing two functions becomes a matter of comparing two angles rather than two line lengths, which has been evaluated as a more difficult perceptual task.¹⁰

Flame charts are a similar code-path visualization to flame graphs (and were inspired by flame graphs¹³). On the *x*-axis, however, they show the passage of time instead of an alphabetical sort. This has its advantages: time-ordered issues can be identified. It can greatly reduce merging, however, a problem exacerbated when profiling multiple threads. It could be a useful option for understanding time order sequences when used with flame graphs for the bigger picture.

Challenges

Challenges with flame graphs mostly involve system profilers and not flame graphs themselves. There are two typical problems with profilers:

- ▶ **Stack traces are incomplete.** Some system profilers truncate to a fixed stack depth (for example, 10 frames), which must be increased to capture the full stack traces, or else frame merging can fail. A worse problem is when the software compiler reuses the frame pointer register as a compiler optimization, breaking the typical method of stack-trace collection. The fix requires either a different compiled binary (for example, using `gcc's -fno-omit-frame-pointer`) or a different stack-walking technique.

- ▶ **Function names are missing.** In this case, the stack trace is complete, but many function names are missing and may be represented as hexadecimal addresses. This commonly happens with JIT (just-in-time) compiled code, which may not create a standard symbol table for profilers. Depending on the profiler and runtime, there are different fixes. For example, Linux `perf_events` supports supplemental symbol files, which the application can create.

At Netflix we encountered both problems when attempting to cre-

ate flame graphs for Java.⁶ The first has been fixed by the addition of a JVM (Java Virtual Machine) option—`XX:+PreserveFramePointer`, which allows Linux `perf_events` to capture full stack traces. The second has been fixed using a Java agent, `perf-map-agent`,¹¹ which creates a symbol table for Java methods.

One challenge with the Perl flame-graph implementation has been the resulting SVG file size. For a large profile with many thousands of unique code paths, the SVG file can be tens of megabytes in size, which becomes sluggish to load in a browser. The fix has been to elide code paths that are so thin they are normally invisible in the flame graph. This does not affect the

big-picture view and has kept the SVG file smaller.

Other Color Schemes

Apart from a random warm palette, other flame-graph color schemes can be used, such as for differentiating code or including an extra dimension of data.

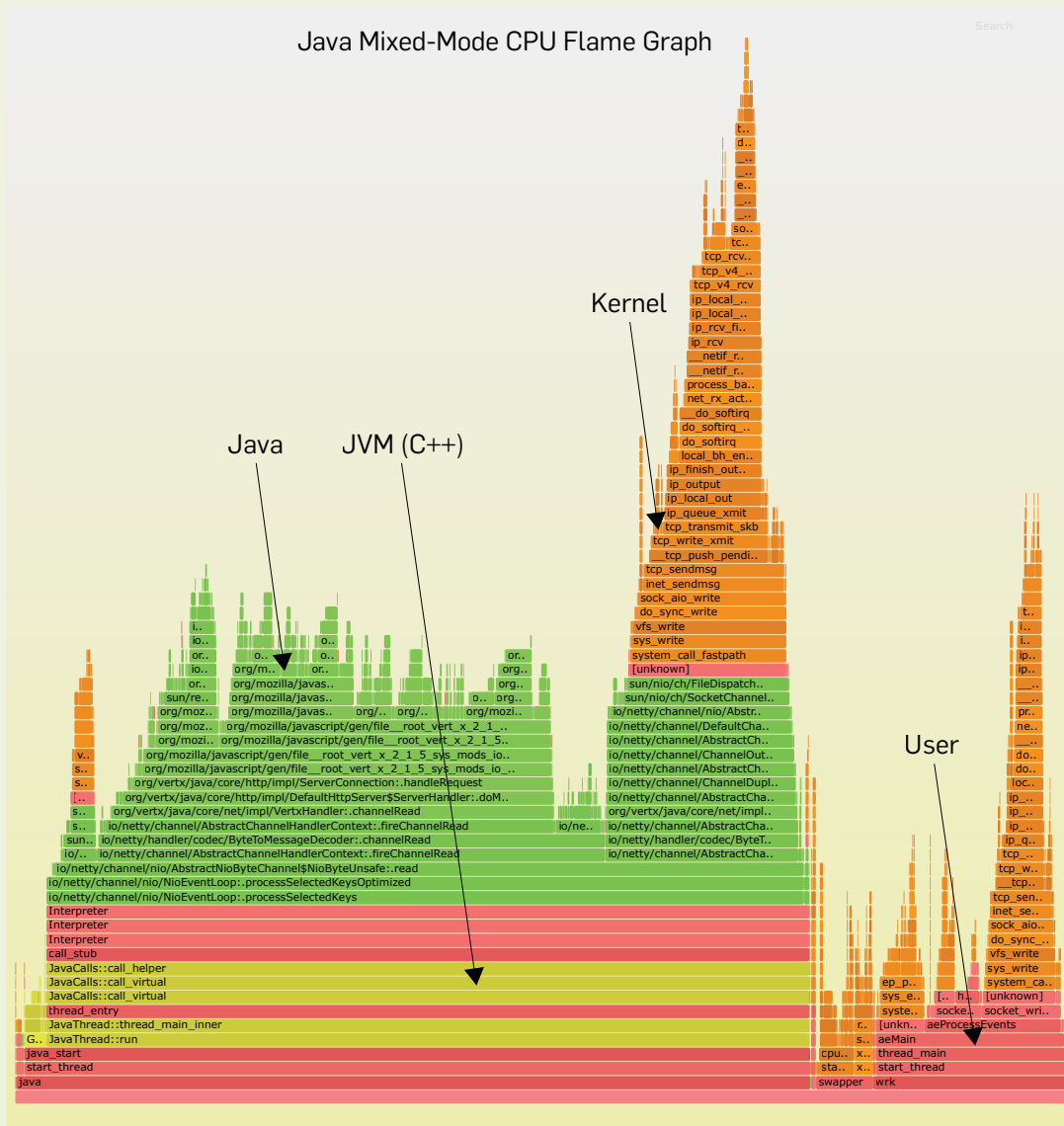
Various palettes can be selected in the Perl flame-graph version, including “java,” which uses different hues to highlight a Java mixed-mode flame graph: green for Java methods, yellow for C++, red for all other user-mode functions, and orange for kernel-mode functions. An example is shown in Figure 7. (An interactive version of Figure 7 in SVG format is available at <http://queue.acm.org/downloads/2016/Gregg7.svg>.)

Another option is a hashing color scheme, which picks a color based on a hash of the function name. This keeps colors consistent, which is helpful when comparing multiple flame graphs from the same system.

Differential Flame Graphs

A differential flame graph shows the difference between two profiles, A and B. The Perl flame-graph software currently supports one method, where the B profile is displayed and then colored using the delta from A to B. Red colors indicate functions that increased, and blue colors indicate those that decreased. A problem with this approach is that some code paths present in the A profile may be miss-

Figure 7. Java mixed-mode CPU flame graph.



ing entirely in the B profile, and so will be missing from the final visualization. This could be misleading.

Another implementation, `flame-graphdiff`,² solves this problem by using three flame graphs. The first shows the A profile, the second shows the B profile, and the third shows only the delta between them. A mouse-over of one function in any flame graph also highlights the others to help navigation. Optionally, the flame graphs can also be colored using a red/blue scheme to indicate which code paths increased or decreased.

Other Targets

As previously mentioned, flame graphs can visualize any profiler output. This includes stack traces collected on CPU PMC (performance monitoring counter) overflow events, static tracing events, and dynamic tracing events. Following are some specific examples.

Stall cycles. A stall-cycle flame graph shows code paths that commonly block on processor or hardware resources—typically memory I/O. The input stack traces can be collected using a PMC profiler, such as Linux `perf` events. This can direct the developer to employ a different optimization technique to the identified code paths, one that aims to reduce memory I/O rather than reducing instructions.

CPI (cycles per instruction), or its invert, IPC (instructions per cycle), is a measure that also helps explain the types of CPU cycles and can direct tuning effort. A CPI flame graph shows a CPU sample flame graph where widths correspond to CPU cycles, but it uses a color scale from red to blue to indicate each function's CPI: red for a high CPI and blue for a low CPI. This can be accomplished by capturing two profiles—a CPU sample profile and an instruction count profile—and then using a differential flame graph to color the difference between them.

Memory. Flame graphs can shed light on memory growth by visualizing a number of different memory events.

A `malloc()` flame graph, created by tracing the `malloc()` function, visualizes code paths that allocated memory. This can be difficult in practice, as allocator functions can be called frequently, making the cost to trace them prohibitive in some scenarios.



The problem that led to the creation of flame graphs was the study of application performance in the cloud.



Tracing the `brk()` and `mmap()` syscalls can show code paths that caused an expansion in virtual memory for a process, typically related to the allocation path, although this could also be an asynchronous expansion of the application's memory. These are typically lower frequency, making them more suitable for tracing.

Tracing memory page faults shows code paths that caused an expansion in physical memory for a process. Unlike allocator code paths, this shows the code that populated the allocated memory. Page faults are also typically a lower-frequency activity.

I/O. The issuing of I/O, such as file system, storage device, and network, can usually be traced using system tracers. A flame graph of these profiles illustrates different application paths that synchronously issued I/O.

In practice, this has revealed types of I/O that were otherwise not known. For example, disk I/O may be issued: synchronously by the application, by a file system read-ahead routine, by an asynchronous flush of dirty data, or by a kernel background scrub of disk blocks. An I/O flame graph identifies each of these types by illustrating the code paths that led to issuing disk I/O.

Off-CPU. Many performance issues are not visible using CPU flame graphs, as they involve time spent while the threads are blocked, not running on a CPU (off-CPU). Reasons for a thread to block include waiting on I/O, locks, timers, a turn on-CPU, and waiting for paging or swapping. These scenarios can be identified by the stack trace when the thread was descheduled. The time spent off-CPU can also be measured by tracing the time from when a thread left the CPU to when it returned. System profilers commonly use static trace points in the kernel to trace these events.

An off-CPU time flame graph can illustrate this off-CPU time by showing the blocked stack traces where the width of a box is proportional to the time spent blocked.

Wakeups. A problem found in practice with off-CPU time flame graphs is they are inconclusive when a thread blocks on a conditional variable. We needed information on why the conditional variable was held by some other thread for so long.

A wakeup time flame graph can be generated by tracing thread wakeup events. This includes wakeups by the other threads releasing the conditional variable, and so they shed light on why they were blocked. This flame-graph type can be studied along with an off-CPU time flame graph for more information on blocked threads.

Chain graphs. One wakeup flame graph may not be enough. The thread that held a conditional variable may have been blocked on another conditional variable, held by another thread. In practice, one thread may have been blocked on a second, which was blocked on a third, and a fourth.

A chain flame graph is an experimental visualization³ that begins with an off-CPU flame graph and then adds all wakeup stack traces to the top of each blocked stack. By reading bottom up, you see the blocked off-CPU stack trace, and then the first stack trace that woke it, then the next stack trace that woke it, and so on. Widths correspond to the time that threads were off-CPU and the time taken for wakeups.

This can be accomplished by tracing all off-CPU and wakeup events with time stamps and stack traces, and post processing. These events can be extremely frequent, however, and impractical to instrument in production using current tools.

Future Work

Much of the work related to flame graphs has involved getting different profilers to work with different runtimes so the input for flame graphs can be captured correctly (for example, for Node.js, Ruby, Perl, Lua, Erlang, Python, Java, golang, and with DTrace, perf_events, pmcstat, Xperf, Instruments, among others). There is likely to be more of this type of work in the future.

Another in-progress differential flame graph, called a white/black differential, uses the single flame-graph scheme described earlier plus an extra region on the right to show only the missing code paths. Differential flame graphs (of any type) should also see more adoption in the future; at Netflix, we are working to have these generated nightly for microservices: to identify regressions and aid with performance-issue analysis.

Several other flame-graph implementations are in development, exploring different features. Netflix has been developing d3-flame-graph,¹² which includes transitions when zooming. The hope is that this can provide new interactivity features, including a way to toggle the merge order from bottom-up to top-down, and also to merge around a given function. Changing the merge order has already proven useful for the original flame-graph.pl, which can optionally merge top-down and then show this as an icicle plot. A top-down merge groups together leaf paths, such as spin locks.

Conclusion

The flame graph is an effective visualization for collected stack traces and is suitable for CPU profiling, as well as many other profile types. It creates a visual map for the execution of software and allows the user to navigate to areas of interest. Unlike other code-path visualizations, flame graphs convey information intuitively using line lengths and can handle large-scale profiles, while usually remaining readable on one screen. The flame graph has become an essential tool for understanding profiles quickly and has been instrumental in countless performance wins.

Acknowledgments

Inspiration for the general layout, SVG output, and JavaScript interactivity came from Neelakanth Nadgir's function_call_graph.rb time-ordered visualization for callstacks,⁹ which itself was inspired by Roch Bourbonnais's CallStackAnalyzer and Jan Boerhout's vfttrace. Adrien Mahieux developed the horizontal zoom feature for flame graphs, and Thorsten Lorenz added a search feature to his implementation.⁸ Cor-Paul Bezemer researched differential flame graphs and developed the first solution.¹ Off-CPU time flame graphs were first discussed and documented by Yichun Zhang.¹⁵

Thanks to the many others who have documented case studies, contributed ideas and code, given talks, created new implementations, and fixed profilers to make this possible. See the updates section for a list of this work.⁵ Finally, thanks to Deirdré Straughan for editing and feedback.

Related articles on queue.acm.org

Interactive Dynamics for Visual Analysis

Ben Shneiderman

<http://queue.acm.org/detail.cfm?id=2146416>

The Antifragile Organization

Ariel Tseitlin

<http://queue.acm.org/detail.cfm?id=2499552>

JavaScript and the Netflix User Interface

Alex Liu

<http://queue.acm.org/detail.cfm?id=2677720>

References

1. Bezemer, C.-P. Flamegraphdiff. GitHub; <http://corpaul.github.io/flamegraphdiff/>.
2. Bezemer, C.-P., Pouwelse, J., Gregg, B. Understanding software performance regressions using differential flame graphs. Published in *IEEE 22nd International Conference on Software Analysis, Evolution and Reengineering* (2015): http://ieeexplore.ieee.org/xpl/login.jsp?tp=&number=7081872&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxppls%2Fabs_all.jsp%3Farnumber%3D7081872.
3. Gregg, B. Blazing performance with flame graphs. In *Proceedings of the 27th Large Installation System Administration Conference* (2013); <https://www.usenix.org/conference/lisa13/technical-sessions/plenary/gregg>.
4. Gregg, B. FlameGraph. GitHub; <https://github.com/brendangregg/FlameGraph>.
5. Gregg, B. Flame graphs; <http://www.brendangregg.com/flamegraphs.html>.
6. Gregg, B., Spier, M. Java in flames. The Netflix Tech Blog, 2015; <http://techblog.netflix.com/2015/07/java-in-flames.html>.
7. Heer, J., Bostock, M., Ogievetsky, V. A tour through the visualization zoo. *acmqueue* 8, 5 (2010); <http://queue.acm.org/detail.cfm?id=1805128>.
8. Lorenz, T. Flamegraph. GitHub; <https://github.com/thlorenz/flamegraph>.
9. Nadgir, N. Visualizing callgraphs via dtrace and ruby. Oracle Blogs, 2007; https://blogs.oracle.com/realneel/entry/visualizing_callstacks_via_dtrace_and_ruby.
10. Odds, G. The science behind data visualisation. Creative Bloq, 2013; <http://www.creativebloq.com/design/science-behind-data-visualisation-8135496>.
11. Rudolph, J. perf-map-agent. GitHub; <https://github.com/jrudolph/perf-map-agent>.
12. Spier, M. d3-flame-graph. GitHub, 2015; <https://github.com/spiermar/d3-flame-graph>.
13. Tikhonovskiy, I. Web Inspector: implement flame chart for CPU profiler. Webkit Bugzilla, 2013; https://bugs.webkit.org/show_bug.cgi?id=111162.
14. Weidendorfer, J. KCachegrind; <https://kcachegrind.github.io/html/Home.html>.
15. Zhang, Y. Introduction to off-CPU time flame graphs, 2013; <http://agentzh.org/misc/slides/off-cpu-flame-graphs.pdf>.

Brendan Gregg is a senior performance architect at Netflix, where he does large-scale computer performance design, analysis, and tuning. He was previously a performance lead and kernel engineer at Sun Microsystems. His recent work includes developing methodologies and visualizations for performance analysis.

Farsighted physicists of yore were danged smart!

BY PAT HELLAND

Standing on Distributed Shoulders of Giants

If you squint hard enough, many of the challenges of distributed computing appear similar to the work done by the great physicists. Dang, those fellows were smart!

Here, I examine some of the most important physics breakthroughs and draw some whimsical parallels to phenomena in the world of computing ... just for fun.

Newton Thought He Knew What Time It Was

Isaac Newton (1642–1727) was a brilliant physicist who defined the foundations for classical mechanics, laws of motion, and universal gravitation. He also built the first refracting telescope, developed a theory of color, and much more. He was one bad dude.

Newton saw the notion of time as constant and consistent across the universe. Furthermore, he assumed that gravity operated instantaneously without regard to distance. Each object in the universe is exerting gravitational force at all times.

This is very much like what we see in a single computer or in a tightly coupled cluster of computers that perform consistent work in a shared transaction. Transactions have a clearly defined local notion of time. Each transaction sees its work as crisply following a set of transactions. Time marches forward unperturbed by distance.

When I was studying computer science (and Nixon was president), we thought about only one computer. There was barely any network other than the one connecting terminals to the single computer. Sometimes, a tape would arrive from another computer and we had to figure out how to understand the data on it. We never thought much about time across computers. It would take a few years before we realized our perspective was too narrow.

Einstein Had Many Watches

In 1905, Albert Einstein (1879–1955) proposed the special theory of relativity based on two principles. First, the laws of physics, including time, appear to be the same to all observers. Second, the speed of light is unchanging.

An implication of this theory is that there is no notion of simultaneity. The notion of simultaneity is relative to the observer, and the march of time is also relative to the observer. Each of these frames of reference is separated by the speed of light as interpreted relative to their speed in space.



This concept has some interesting consequences. The sun might have blown up five minutes ago, and the next three minutes will be lovely. When stuff happens far away, it takes time to find out ... potentially a long time.

In computing, you cannot know what is happening “over there.” Interacting with another system always takes time. You can launch a message, but you always have to wait for the answer to come back to know the result. More and more, latency is becoming the major design point in systems.

The time horizon for knowledge propagation in a distributed system is unpredictable. This is even worse than in the physical Einstein-based universe. At least with our sun and the speed of light, we know we can see what is happening at the sun as of eight minutes ago. In a distributed system, we have a statistical understanding of how our knowledge propagates, but we simply cannot know with certainty. The other server, in its very own time

domain, may be incommunicado for a heck of a long time.

Furthermore, in any distributed interaction, a message may or may not be delivered within bounded time. Higher-level applications don’t ever know if the protocol completed. Figure 1 shows how the last message delivery is not guaranteed and the sender never knows what the receiver knows. In any distributed protocol, the sender of the last message cannot tell whether it arrived. That would require another message.

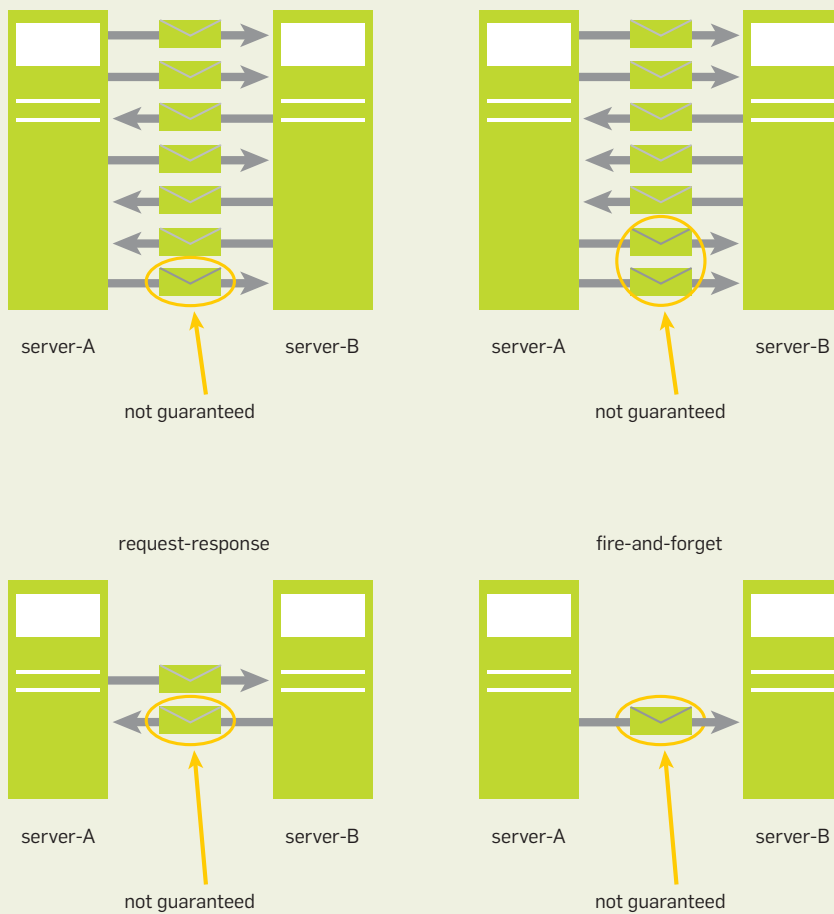
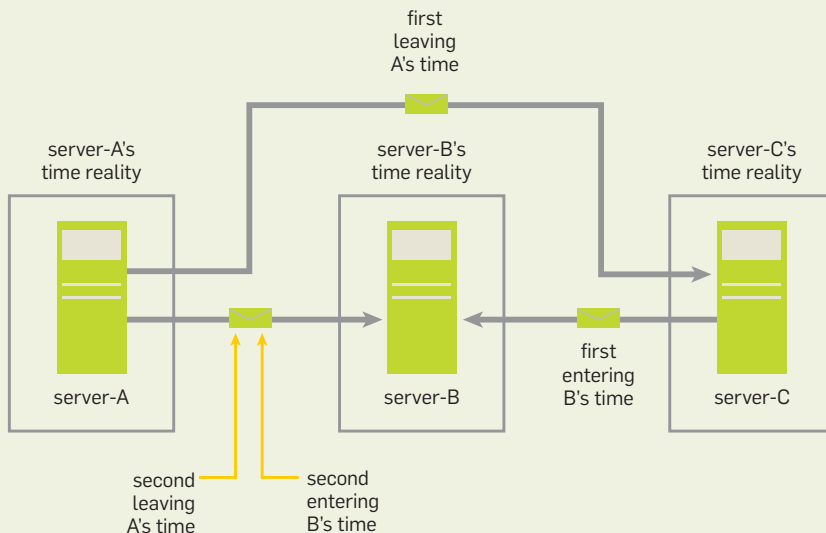
Another problem is that servers and messages live in their very own time space. Messages sent and received across multiple servers may have surprising reorderings. Each server and each message lives in its own time, and they may be relative to each other but may offer surprises because they are not coordinated. Some appear slower, and some faster. This is annoying.

In Figure 2, as work flows across different times in servers and messages,

the time is disconnected and may be slower or faster than expected. In this case, the second message sent by A may arrive after work caused by the first message, traveling through C. These problems can make your head hurt in a similar fashion to how it hurts when contemplating twins where one travels close to the speed of light and time appears to slow down while the other one stays home and ages.

You cannot do distributed agreement in bounded time. Messages get lost. You can retry them and they will probably get through. In a fixed period of time, however, there is a small (perhaps very small) chance they won’t arrive. For any fixed period of time, there’s a chance the partner server will be running *slloooooow* and not get back.

Two-phase commit cannot guarantee agreement in bounded time. Similarly, Paxos,⁷ Raft,⁸ and the other cool agreement protocols cannot guarantee agreement in a bounded time. These protocols are *very likely to reach agree-*

Figure 1. Sender gets no confirmation of final message delivery.**Figure 2. Disconnected time may be slower or faster than expected.**

ment soon, but there's no guarantee.⁴ Each lives in its own relative world and does not know what is happening over there ... at least not yet.

According to the CAP Theorem^{1,5} (that is, consistency, availability, partition tolerance), if you tolerate failures of computers and/or networks, you can have either classic database consistency or database availability. To avoid application challenges, most systems choose consistency over availability.

Two-phase commit is the anti-availability protocol.

From where I stand, Einstein made a lot of sense. I'm not sure how you feel about him.

Hubble Was Increasingly Far Out

Edwin Hubble (1889–1953) was an astronomer who discovered the farther away an object is, the faster it is receding from us. This, in turn, implies the universe is expanding. Basically, everything is getting farther away from everything else.

In computing, we have seen an ever-increasing amount of computation, bandwidth, and memory size. It looks like this will continue for a while. Latency is not decreasing too much and is limited by the speed of light. There are no obvious signs that the speed of light will stop being a constraint anytime soon. The number of instruction opportunities lost to waiting while something is fetched is increasing inexorably.

Computing is like the Hubble's universe ... Everything is getting farther away from everything else.

Shared read-only data isn't the biggest problem. With enough cache, you can pull the stuff you need into the sharing system. Sharing writeable stuff is a disaster. You frequently stall while pulling a cache line with the latest copy from a cohort's cache. More and more instruction opportunities will be lost while waiting. *This will only get worse as time moves on!*

Shared memory works great ... as long as you don't SHARE memory.

Either we figure out how to get around that pesky speed-of-light thing, or we are going to need to work harder on asynchrony and concurrency.

Heisenberg Wasn't Sure

Werner Heisenberg (1901–1976) defined the uncertainty principle, which states that the more you know about the location of a particle, the less you know about its movement. Basically, you can't know everything about anything.

In a distributed system you have a gaggle of servers, each of which lives in various states of health, death, or garbage collection. The vast majority of the time you can chat with a server and get a crisp and timely result. Other times you do not get a prompt answer and it's difficult to know if you should abandon the slacker or wait patiently. Furthermore, you don't know if the server got the request, did the work, and just has not answered. Anytime a request goes to a single system, you don't know when the request will be delayed.^{2,6}

In some distributed systems, it is essential to have an extremely consistent and fast response time for online users. To accomplish this, multiple requests must be issued, and the completion of a subset of the requests is accepted as happiness. In a distributed system, you can know where the work is done or you can know when the work is done but you can't know both.

To know *when* a request is done within a statistical SLA (service-level agreement), you need to accept that you do not know *where* the work will be done. Retries of the request are the only option to get a timely answer often enough. Hence, the requests had better be idempotent.

Schrödinger's PUT

Erwin Schrödinger (1887–1961) was a leading physicist of the early 20th century. While he made many substantial contributions to field quantum theory, he is most often remembered for a thought experiment designed to show the challenges of quantum physics.

In quantum physics the theory, the math, and the experimental observations show that pretty much everything remains in multiple states until it in-

teracts with or is observed by the external world. This is known as a superposition of states that collapse when you actually look.

To show this seems goofy, Schrödinger proposed this quantum-level uncertainty could map to a macro-level uncertainty. Start by placing a tiny bit of uranium, a Geiger counter, a vial of cyanide, and a cat into a steel box. Rig the Geiger counter to use a hammer to break the vial of cyanide if an atom of uranium has decayed. Since the quantum physics of uranium decay show it is both decayed and not decayed until you observe the state, it is clear the cat is both simultaneously dead and alive. Turns out many contemporary physicists think it's not goofy ... the cat would be in both states. Go figure!

New distributed systems such as Dynamo³ store their data in unpredictable locations. This allows prompt and consistent latencies for PUTs as well as self-managing and self-balancing servers. Typically, the client issues a PUT to each of three servers, and when the cluster is automatically rebalancing, the destination servers may be sloshing data around. The set of servers used as destinations may be slippery. A subsequent GET may need to try many servers to track down the new value. If a client dies during a PUT, it is possible that no servers received the new value or that only a single server received it. That single server may or may not die before sharing the news. That single server may die, not be around to answer a read, and then later pop back to life resurrecting the missing PUT.

Therefore, a subsequent GET may find the PUT, or it may not. There is effectively no limit to the number of places it may be hiding. There is no upper bound on the time taken for the new value to appear. If it does appear, it will be re-replicated to make it stick.

*While not yet observed,
a PUT does not really exist ...
it's likely to exist but you can't be sure.
Only after it is seen by a GET
will the PUT really exist.*

Furthermore, the failure to observe does not mean the PUT is really missing. It may be lurking in a dead or unresponsive machine. If you see the PUT and force its replication to multiple

servers, it remains in existence with very high fidelity. Not seeing it tells you only that it's likely it is not there.

Conclusion

Wow! There have been lots of brilliant physicists, many of them not mentioned here. Much of their work has shown us the very counterintuitive ways the world works. Year after year, there are new understandings and many surprises.

In our nascent discipline of distributed systems, we would be wise to realize there are subtleties, surprises, and bizarre uncertainties intrinsic in what we do. Understanding, bounding, and managing the trade-offs inherent in these systems will be a source of great challenge for years to come. I think it's a lot of fun! C

Related articles on queue.acm.org

As Big as a Barn?

Stan Kelly-Bootle

<http://queue.acm.org/detail.cfm?id=1229919>

Condos and Clouds

Pat Helland

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

Testable System Administration

Mark Burgess

<http://queue.acm.org/detail.cfm?id=1937179>

References

1. Brewer, E.A. Towards robust distributed systems. In *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing* (2000).
2. Dean, J., Barroso, L.A. 2013. The tail at scale. *Commun. ACM* 56, 2 (Feb. 2013), 74–80.
3. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W. Dynamo: Amazon's highly available key-value store. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles* (2007), 205–220.
4. Fischer, M., Lynch, N., Paterson, M. The impossibility of distributed consensus with one faulty process. *JACM* 32, 2 (Apr. 1985).
5. Gilbert, S., Lynch, N. Brewer's conjecture and the feasibility of consistent, available, and partition-tolerant web services. *ACM SIGACT News* 33, 2 (2002).
6. Helland, P. Heisenberg was on the write track. In *Proceedings of the 7th Biennial Conference on Innovative Data Systems Research* (2015).
7. Lamport, L. The part-time parliament. *ACM Trans. Computer Systems* 16, 2 (May 1998).
8. Ongaro, D., Ousterhout, J. In search of an understandable consensus algorithm. In *Proceedings of the Usenix Annual Technical Conference* (2014); <https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro>.

Pat Helland has been implementing transaction systems, databases, application platforms, distributed systems, fault-tolerant systems, and messaging systems since 1978. He currently works at Salesforce.

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

DOI:10.1145/2896587

Human-centered design can make application programming interfaces easier for developers to use.

BY BRAD A. MYERS AND JEFFREY STYLOS

Improving API Usability

APPLICATION PROGRAMMING INTERFACES (APIs), including libraries, frameworks, toolkits, and software development kits, are used by virtually all code. If one includes both internal APIs (interfaces internal to software projects) and public APIs (such as the Java Platform SDK, the Windows .NET Framework, jQuery for JavaScript, and Web services like Google Maps), nearly every line of code most programmers write will use API calls. APIs provide a mechanism for code reuse so programmers can build on top of what others (or they themselves) have already done, rather than start from scratch with every program. Moreover, using APIs is often required because low-level access to system resources (such as graphics, networking, and the file system) is available only through protected APIs. Organizations increasingly provide their internal data on the Web through public APIs; for example, <http://www.programmableweb.com> lists almost 15,000 APIs for Web services and <https://www.digitalgov.gov/2013/04/30/apis-in-government/> promotes use of government data through Web APIs.

There is an expanding market of companies, software, and services to help organizations provide APIs. One such company, Apigee Corporation (<http://apigee.com/>), surveyed 200 marketing and IT executives in U.S. companies with annual revenue of more than \$500 million in 2013, with 77% of respondents rating APIs “important” to making their systems and data available to other companies, and only 1% of respondents rating APIs as “not at all important.”¹² Apigee estimated the total market for API Web middleware was \$5.5 billion in 2014.

However, APIs are often difficult to use, and programmers at all levels, from novices to experts, repeatedly spend significant time learning new APIs. APIs are also often used incorrectly, resulting in bugs and sometimes significant security problems.⁷ APIs must provide the needed functionality, but even when they do, the design could make them unusable. Because APIs serve as the interface between human developers and the body of code that implements the functionality, principles and methods from human-computer interaction (HCI) can be applied to improve usability. “Usability,” as discussed here, includes a variety of properties, not just learnability for developers unfamiliar with an API but also efficiency and correctness when used by experts. This property is sometimes called “DevX,” or developer experience, as an analogy with “UX,” or user experience. But usability also includes providing the appropriate functionality and ways to access it. Researchers have shown how various

» key insights

- All modern software makes heavy use of APIs, yet programmers can find APIs difficult to use, resulting in errors and inefficiencies.
- A variety of research findings, tools, and methods are widely available for improving API usability.
- Evaluating and designing APIs with their users in mind can result in fewer errors, along with greater efficiency, effectiveness, and security.



human-centered techniques, including contextual inquiry field studies, corpus studies, laboratory user studies, and logs from field trials, can be used to determine the actual requirements for APIs so they provide the right functionality.²¹ Other research focuses on access to that functionality, showing, for example, software patterns in APIs that are problematic for users,^{6,10,25} guidelines that can be used to evaluate API designs,^{4,8} with some assessed by

automated tools,^{18,20} and mitigations to improve usability when other considerations require trade-offs.^{15,23} As an example, our own small lab study in 2008 found API users were between 2.4 and 11.2 times faster when a method was on the expected class, rather than on a different class.²⁵ Note we are not arguing usability should always overshadow other considerations when designing an API; rather, API designers should add usability as explicit design-

and-evaluation criteria so they do not create an unusable API inadvertently, and when they intentionally decrease usability in favor of some other criteria, at least to do it knowingly and provide mitigations, including specific documentation and tool support.

Developers have been designing APIs for decades, but without empirical research on API usability, many of them have been difficult to use, and some well-intentioned design recom-

mentations have turned out to be wrong. There was scattered interest in API usability in the late 1990s, with the first significant research in the area appearing in the first decade of the 2000s, especially from the Microsoft Visual Studio usability group.⁴ This resulted in a gathering of like-minded researchers who in 2009 created the API Usability website (<http://www.apiusability.org>) that continues to be a repository for API-usability information.

We want to make clear the various stakeholders affected by APIs. The first is API designers, including all the people involved in creating the API, like API implementers and API documentation writers. Some of their goals are to maximize adoption of an API, minimize support costs, minimize development costs, and release the API in a timely fashion. Next is the API users, or the programmers who use APIs to help them write their code. Their goals include being able to quickly write error-free programs (without having to limit their scope or features), use APIs many other programmers use (so others can test them, answer questions, and post sample code using the APIs), not needing to update their code due to changes in APIs, and having their resulting applications run quickly and efficiently. For public APIs, there may be thousands of times as many API users as there are API developers. Finally, there are the consumers of the resulting products who may be indirectly affected by the quality of the resulting code but who also might be directly affected, as in, say, the case of user-interface widgets, where API choices affect the look and feel of the resulting user interface. Consumers' goals include having products with the desired features, robustness, and ease of use.

Motivating the Problem

One reason API design is such a challenge is there are many quality attributes on which APIs might be evaluated for the stakeholders (see Figure 1), as well as trade-offs among them. At the highest level, the two basic qualities of an API are usability and power. Usability includes such attributes as how easy an API is to learn, how productive programmers are using it, how



APIs are also often used incorrectly, resulting in bugs and sometimes significant security problems.



well it prevents errors, how simple it is to use, how consistent it is, and how well it matches its users' mental models. Power includes an API's expressiveness, or the kinds of abstractions it provides; its extensibility (how users can extend it to create convenient user-specific components); its "evolvability" for the designers who will update it and create new versions; its performance in terms of speed, memory, and other resource consumption; and the robustness and security of its implementation and resulting application. Usability mostly affects API users, though error prevention also affects consumers of the resulting products. Power affects mostly API users and product consumers, though evolvability also affects API designers and, indirectly, API users to the extent changes in the API require editing the code of applications that use it. Modern APIs for Web services seem to involve such "breaking changes" more than desktop APIs, as when, say, migrating from v2 to v3 of the Google Maps API required a complete rewrite of the API users' code. We have heard anecdotal evidence that usability can also affect API adoption; if an API takes too long for a programmer to learn, some organizations choose to use a different API or write simpler functionality from scratch.

Another reason for difficulty is the design of an API requires making hundreds of design decisions at many different levels, all of which can affect usability.²⁴ Decisions range from the global (such as the overall architecture of the API, what design patterns will be used, and how functionality will be presented and organized) down to the low level (such as specific name of each exported class, function, method, exception, and parameter). The enormous size of public APIs contributes to these difficulties; for example, the Java Platform, Standard Edition API Specification includes more than 4,000 classes with more than 35,000 different methods, and Microsoft's .NET Framework includes more than 140,000 classes, methods, properties, and fields.

Examples of Problems

All programmers are likely able to identify APIs they personally had difficulty learning and using correctly due to us-

ability limitations.^a We list several examples here to give an idea of the range of problems. Other publications have also surveyed the area.^{10,24}

Studies of novice programmers have identified selecting the right facilities to use, then understanding how to coordinate multiple elements of APIs as key barriers to learning.¹³ For example, in Visual Basic, learners wanted to “pull” data from a dialogue box into a window after “OK” was hit, but because controls are inaccessible if their dialogue box is not visible in Visual Basic, data must instead be “pushed” from the dialogue to the window.

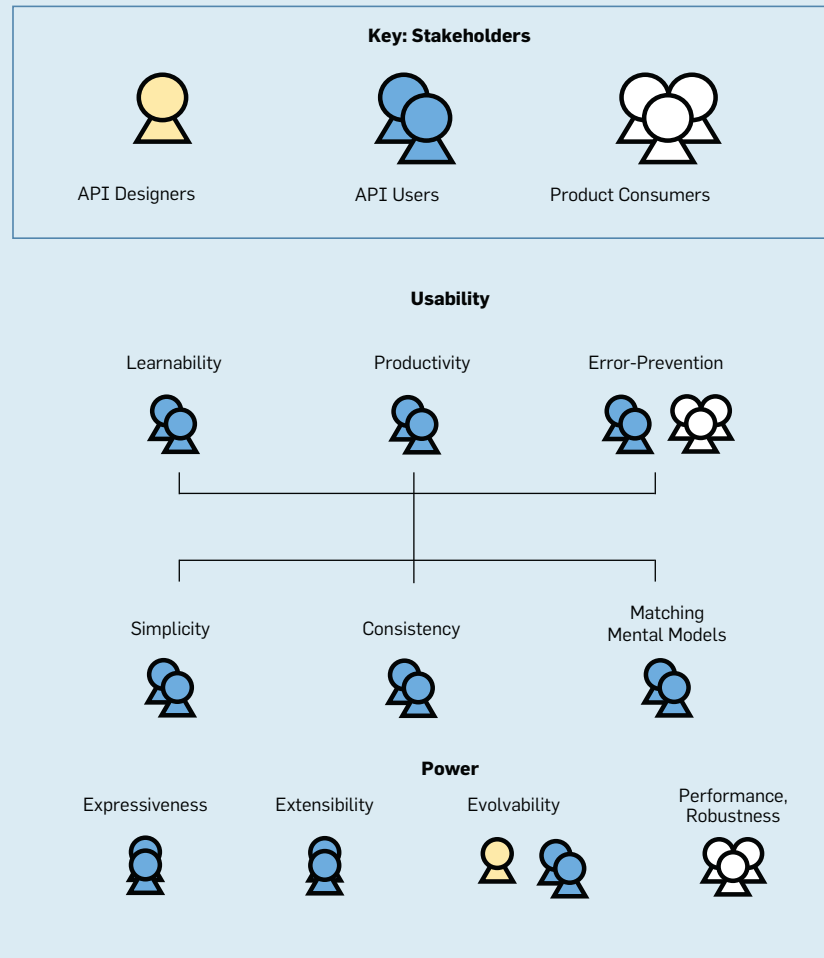
There are many examples of API quirks affecting expert professional programmers as well. For example, one study¹¹ detailed a number of functionality and usability problems with the .NET socket `Select()` function in C#, using it to motivate greater focus on the usability of APIs in general. In another study,²¹ API users reported difficulty with SAP’s BRFplus API (a business-rules engine), and a redesign of the API dramatically improved users’ success and time to completion. A study of the early version of SAP’s APIs for enterprise Service-Oriented Architecture, or eSOA,¹ identified problems with documentation, as well as additional weaknesses with the API itself, including names that were too long (see Figure 2), unclear dependencies, difficulty coordinating multiple objects, and poor error messages when API users made mistakes. Severe problems with documentation

were also highlighted by a field study¹⁹ of 440 professional developers learning to use Microsoft’s APIs.

Many sources of API recommendations are available in print and online. Two of the most comprehensive are books by Joshua Bloch (then at Sun Microsystems)³ and by Krzysztof Cwalina and Brad Abrams (then at Microsoft). Each offers guidelines devel-

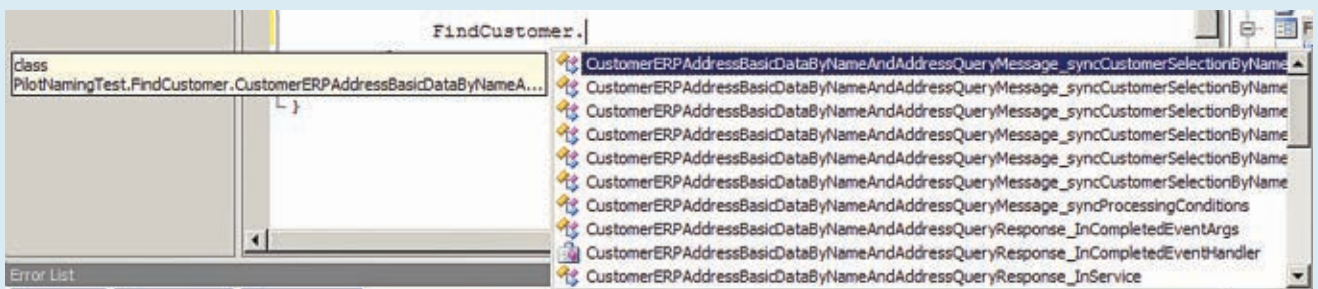
oped over several years during creation of such widespread APIs as the Java Development Kit and the .NET base libraries, respectively. However, we have found some of these guidelines to be contradicted by empirical evidence. For example, Bloch discussed the many architectural advantages of the factory pattern,⁹ where objects in a class-instance object system cannot

Figure 1. API quality attributes and the stakeholders most affected by each quality.



^a We are collecting a list of usability concerns and problems with APIs; please send yours to author Brad A. Myers; for a more complete list of articles and resources on API usability, see <http://www.apiusability.org>

Figure 2. Method names are so long users cannot tell which of the six methods to select in autocomplete;¹ note the autocomplete menu does not support horizontal scrolling nor does the yellow hover text for the selected item.



be created by calling new but must instead be created using a separate “factory” method or entirely different factory class. Use of other patterns (such as the singleton or flyweight patterns)⁹ could also require factory methods. However, empirical research has shown significant usability penalties when using the factory pattern in APIs.⁶

There is also plenty of evidence that less usable API designs affect security. Increasing API usability often increases security. For example, a study by Fahl et al.⁷ of 13,500 popular free Android apps found 8.0% had misused the APIs for the Secure Sockets Layer (SSL) or its successor, the Transport Layer Security (TLS), and were thus vulnerable to man-in-the-middle and other attacks; a follow-on study of Apple iOS apps found 9.7% to be vulnerable. Causes include significant difficulties using security APIs correctly, and Fahl et al.⁷ recommended numerous changes that would increase the usability and security of the APIs.

On the other hand, increased security in some cases seems to lower usability of the API. For example, Java security guidelines strongly encourage classes that are immutable, meaning objects cannot be changed after they are constructed.¹⁷ However, empirical research shows professionals trying to learn APIs prefer to be able to create empty objects and set their fields later, thus requiring mutable classes.²² This programmer preference illustrates that API design

involves trade-offs and how useful it is to know what factors can influence usability and security.

Human-Centered Methods

If you are convinced API usability should be improved, you might wonder how it can be done. Fortunately, a variety of human-centered methods are available to help answer the questions an API designer might have.

Design phase. At the beginning of the process, as an API is being planned, many methods can help the API designer. The Natural Programming Project at Carnegie Mellon University has pioneered what we call the “natural programming” elicitation method, where we try to understand how API users are thinking about functionality²⁵ to determine what would be the most natural way to provide it. The essence of this approach is to describe the required functionality to the API users, then ask them to write onto blank paper (or a blank screen) the design for the API. The key goals are to understand the names API users assign to the various entities and how users organize the functionality into different classes, where necessary. Multiple researchers have reported trying to guess the names of classes and methods is the key way users search and browse for the needed functionality,¹⁴ and we have found surprising consistency in how they name and organize the functionality among the classes.²⁵ This elicitation technique also turns out to be useful as part of a usability evaluation of an existing API (described later), as

it helps explain the results by revealing participants’ mental models.

Only a few empirical studies have covered API design patterns but consistently show simplifying the API and avoiding patterns like the factory pattern will improve usability.⁶ Other recommendations on designs are based on the opinions of experienced designers,^{3,5,11,17} though there are many recommendations, and they are sometimes contradictory.

As described here, there is a wide variety of evaluation methods for designs, but many of them can also be used during the design phase as guidelines the API designer should keep in mind. For example, one guideline that appears in “cognitive dimensions”⁴ and in Nielsen’s “heuristic evaluation”¹⁶ is consistency, which applies to many aspects of an API design. One example of its application is that the order of parameters should be the same in every method. However, `javax.xml.stream.XMLStreamWriter` for Java 8 has different overloads for the `writeStartElement` method, taking the `String` parameters `localName` and `namespaceURI` in the opposite order from each other,¹⁸ and, since both are strings, the compiler is not able to detect user errors (see code section 1).

Another Nielsen guideline is to reduce error proneness.¹⁶ It can apply to avoiding long sequences of parameters of the same type the API user is likely to get wrong and the compiler will also not be able to check. For example, the class `TPASupplierOrderXDE` in Petstore (J2EE demonstration software from Oracle) takes a sequence of nine `Strings` (see code section 2).¹⁸

Likewise, in Microsoft’s .Net, `System.Net.Cookie` has four constructors that take zero, two, three, or four strings as input. Another application of this principle is to make the default or example parameters do the right thing. Fahl et al.⁷ reported that, by default, SSL certificate validation is turned off when using some iOS frameworks and libraries, resulting in API users making the error of leaving them unchecked in deployed applications.

Evaluating the API Design

Following its design, a new API should be evaluated to measure and

Code section 1. Two overloads of the `writeStartElement` method in Java where `localName` and `namespaceURI` are in the opposite order.

```
void writeStartElement(String namespaceURI,
    String localName)
void writeStartElement(String prefix,
    String localName,
    String namespaceURI)
```

Code section 2. String parameters many API users are likely to get wrong.

```
void setShippingAddress (
    String firstName, String lastName, String street,
    String city, String state, String country,
    String zipCode, String email, String phone)
```

improve its usability, with a wide variety of user-centered methods available for the evaluation.

The easiest is to evaluate the design based on a set of guidelines. Nielsen's "heuristic evaluation" guidelines¹⁶ describe 10 properties an expert can use to check any design (<http://www.nngroup.com/articles/ten-usability-heuristics/>) that apply equally well to APIs as to regular user interfaces. Here are our mappings of the guidelines to API designs with a general example of how each can be applied.

Visibility of system status. It should be easy for the API user to check the state (such as whether a file is open or not), and mismatches between the state and operations should provide appropriate feedback (such as writing to a closed file should result in a helpful error message);

Match between system and real world. Names given to methods and the organization of methods into classes should match the API users' expectations. For example, the most generic and well-known name should be used for the class programmers are supposed to actually use, but this is violated by Java in many places. There is a class in Java called `File`, but it is a high-level abstract class to represent file system paths, and API users must use a completely different class (such as `FileOutputStream`) for reading and writing;

User control and freedom. API users should be able to abort or reset operations and easily get the API back to a normal state;

Consistency and standards. All parts of the design should be consistent throughout the API, as discussed earlier;

Error prevention. The API should guide the user into using the API correctly, including having defaults that do the right thing;

Recognition rather than recall. As discussed in the following paragraphs, a favorite tool of API users to explore an API is the autocomplete popup from the integrated development environment (IDE), so one requirement is to make the names clear and understandable, enabling users to recognize which element they want. One noteworthy violation of this principle was an API where six names all looked identical in auto-



The most generic and well-known name should be used for the class that programmers are supposed to actually use, but this is violated by Java in many places.



complete because the names were so long the differences were off screen,¹ as in Figure 2. We also found these names were indistinguishable when users were trying to read and understand existing code, leading to much confusion and errors;¹

Flexibility and efficiency of use. Users should be able to accomplish their tasks with the API efficiently;

Aesthetic and minimalist design. It might seem obvious that a smaller and less-complex API is likely to be more usable. One empirical study²⁰ found that for classes, the number of other classes in the same package/namespace had an influence on the success of finding the desired one. However, we found no correlation between the number of elements in an API and its usability, as long as they had appropriate names and were well organized.²⁵ For example, adding more different kinds of objects that can be drawn does not necessarily complicate a graphics package, and adding convenience constructors that take different sets of parameters can improve usability.²⁰ An important factor seems to be having distinct prefixes for the different method names so they are easily differentiated by typing a small number of characters for code completion in the editor;²⁰

Help users recognize, diagnose, and recover from errors. A surprising number of APIs supply unhelpful error information or even none at all when something goes wrong, thus decreasing usability and also possibly affecting correctness and security. Many approaches are available for reporting errors, with little empirical evidence (but lots of opinions) about which is more usable—a topic for our group's current work; and

Help and documentation. A key complaint about API usability is inadequate documentation.¹⁹

Likewise, the Cognitive Dimensions Framework provides a set of guidelines that can be used to evaluate APIs.⁴ A related method is Cognitive Walkthrough² whereby an expert evaluates how well a user interface supports one or more specific tasks. We used both Heuristic Evaluation and Cognitive Walkthrough to help improve the NetWeaver Gateway product from SAP, Inc. Because the SAP

developers who built this tool were using agile software-development processes, they were able to quickly improve the tool's usability based on our evaluations.⁸

Although a user-interface expert usually applies these guidelines to evaluate an API, some tools automate API evaluations using guidelines; for example, one tool can evaluate APIs against a set of nine metrics, including looking for methods that are overloaded but with different return types, too many parameters in a row with the same types, and consistency of parameter orderings across different methods.¹⁸ Likewise, the API Concepts Framework takes the context of use into account, as it evaluates both the API and samples of code using the API.²⁰ It can measure a variety of metrics already mentioned, including whether multiple methods have the same prefix (and thus may be annoying to use in code-completion menus) and use the factory pattern.

Among HCI practitioners, running user studies to test a user interface with target users is considered the “gold standard.”¹⁶ Such user tests can be done with APIs as well. In a think-aloud usability evaluation, target users (here, API users) attempt some tasks (either their own or experimenter-provided) with the API typically in a lab setting and are encouraged to say aloud what they are thinking. This makes clear what they are looking for or trying to achieve and, in general, why they are making certain choices. A researcher might be interested in a more formal A/B test, comparing, say, an old vs. new version of an API (as we previously have done^{6,21,25}), but the insights about usability barriers are usually sufficient when they emerge from an informal think-aloud evaluation.

Grill et al.¹⁰ described a method where they had experts use Nielsen's Heuristic Evaluation to identify problems with an API and observed developers learning to use the same API in the lab. An interesting finding was these two methods revealed mostly independent sets of problems with that API.

Mitigations

When any of these methods reveals a usability problem with an API, an



APIs specify not just the interfaces for programmers to understand and write code against but also for computers to execute, making them brittle and difficult to change.



ideal mitigation would be to change the API to fix the problem. However, actually changing an API may not be possible for a number of reasons. For example, legacy APIs can be changed only rarely since it would involve also changing all the code that uses the APIs. Even with new APIs, an API designer could make an explicit trade-off to decrease usability in favor of other goals, like efficiency. For example, a factory pattern might be used in a performance-critical API to avoid allocating any memory at all.

When a usability problem cannot be removed from the API itself, many mitigations can be applied to help its users. The most obvious is to improve the documentation and example code, which are the subjects of frequent complaints from API users in general.¹⁹ API designers can be careful to explicitly direct users to the solutions to the known problems. For example, the Jadeite tool adds cross-references to the documentation for methods users expect to exist but which are actually in a different class.²³ For example, the Java Message class does not have a send method, so Jadeite adds a pretend send method to the documentation for the Message class, telling users to look in the mail Transport class instead. Knowing users are confused by the lack of this method in the Message class allows API documentation to add help exactly where it is needed.

Tools

This kind of help can be provided even in programming tools (such as the code editor or IDE), not just in the documentation. Calcite¹⁵ adds extra entries into the autocomplete menus of the Eclipse IDE to help API users discover what additional methods will be useful in the current context, even if they are not part of the current class. It also highlights when the factory pattern must be used to create objects.

Many other tools can also help with API usability. For example, some tools that help refactor the API users' code may lower the barrier for changing an API (such as Gofix for the Go language, <http://blog.golang.org/introducing-gofix>). Other tools help find the right elements to use in APIs, “wizards” that produce part

of the needed code based on API users' answers to questions,⁸ and many kinds of bug checkers that check for proper API use (such as <http://find-bugs.sourceforge.net/>).

Conclusion

Since our Natural Programming group began researching API usability in the early 2000s, some significant shifts have occurred in the software industry. One of the biggest is the move toward agile software development, whereby a minimum-viable-product is quickly released and then iterated upon based on real-world user feedback. Though it has had a positive effect on usability overall in driving user-centric development, it exposes some of the unique challenges of API design. APIs specify not just the interfaces for programmers to understand and write code against but also for computers to execute, making them brittle and difficult to change. While human users are nimble responding to the small, gradual changes in user interface design that result from an agile process, code is not. This aversion to change raises the stakes for getting the design right in the first place. API users behave just like other users almost universally, but the constraints created by needing to avoid breaking existing code make the evolution, versioning, and initial release process considerably different from other design tasks. It is not clear how the “fail fast, fail often” style of agile development popular today can be adapted to the creation and evolution of APIs, where the cost of releasing and supporting imperfect APIs or making breaking changes to an existing API—either by supporting multiple versions or by removing support for old versions—is very high.

We envision a future where API designers will always include usability as a key quality metric to be optimized by all APIs and where releasing APIs that have not been evaluated for usability will be as unacceptable as not evaluating APIs for correctness or robustness. When designers decide usability must be compromised in favor of other goals, this decision will be made knowingly, and appropriate mitigations will be put in place. Researchers and API designers will contribute to a body of

knowledge and set of methods and tools that can be used to evaluate and improve API usability. The result will be APIs that are easier to learn and use correctly, API users who are more effective and efficient, and resulting products that are more robust and secure for consumers.

Acknowledgments

This article follows from more than a decade of work on API usability by the Natural Programming group at Carnegie Mellon University by more than 30 students, staff, and postdocs, in addition to the authors, and we thank them all for their contributions. We also thank André Santos, Jack Beaton, Michael Coblenz, John Daughtry, Josh Sunshine, and the reviewers for their comments on earlier drafts of this article. This work has been funded by SAP, Adobe, IBM, Microsoft, and multiple National Science Foundation grants, including CNS-1423054, IIS-1314356, IIS-1116724, IIS-0329090, CCF-0811610, IIS-0757511, and CCR-0324770. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of any of the sponsors. 

References

1. Beaton, J., Jeong, S.Y., Xie, Y., Stylos, J., and Myers, B.A. Usability challenges for enterprise service-oriented architecture APIs. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing* (Herrsching am Ammersee, Germany, Sept. 15–18). IEEE Computer Society Press, Washington, D.C., 2008, 193–196.
2. Blackmon, M.H., Polson, P.G., Kitajima, M., and Lewis, C. Cognitive walkthrough for the Web. In *Proceedings of the Conference on Human Factors in Computing Systems* (Minneapolis, MN, Apr. 20–25). ACM Press, New York, 2002, 463–470.
3. Bloch, J. *Effective Java Programming Language Guide*. Addison-Wesley, Boston, MA, 2001.
4. Clarke, S. *API Usability and the Cognitive Dimensions Framework*, 2003; <http://blogs.msdn.com/stevenc/archive/2003/10/08/57040.aspx>
5. Cwalina, K. and Abrams, B. *Framework Design Guidelines, Conventions, Idioms, and Patterns for Reusable .NET Libraries*. Addison-Wesley, Upper-Saddle River, NJ, 2006.
6. Ellis, B., Stylos, J., and Myers, B.A. The factory pattern in API design: A usability evaluation. In *Proceedings of the International Conference on Software Engineering* (Minneapolis, MN, May 20–26). IEEE Computer Society Press, Washington, D.C., 2007, 302–312.
7. Fahl, S., Harbach, M., Perl, H., Koetter, M., and Smith, M. Rethinking SSL development in an applied world. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security* (Berlin, Germany, Nov. 4–8). ACM Press, New York, 2013, 49–60.
8. Faulring, A., Myers, B.A., Oren, Y., and Rotenberg, K. A case study of using HCI methods to improve tools for programmers. In *Proceedings of Workshop on Cooperative and Human Aspects of Software Engineering at the International Conference on Software Engineering* (Zürich, Switzerland, June 2). IEEE Computer Society Press, Washington, D.C., 2012, 37–39.
9. Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns*. Addison-Wesley, Reading, MA, 1995.
10. Grill, T., Polacek, O., and Tscheligi, M. Methods towards API usability: A structural analysis of usability problem categories. In *Proceedings of the Fourth International Conference on Human-Centered Software Engineering*, M. Winckler et al., Eds. (Toulouse, France, Oct. 29–31). Springer, Berlin, Germany, 2012, 164–180.
11. Henning, M. API design matters. *ACM Queue* 5, 4 (May–June, 2007), 24–36.
12. Kirschner, B. *The Perceived Relevance of APIs*. Apigee Corporation, San Jose, CA, 2015; <http://apigee.com/about/api-best-practices/perceived-relevance-apis>
13. Ko, A.J., Myers, B.A., and Aung, H.H. Six learning barriers in end-user programming systems. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing* (Rome, Italy, Sept. 26–29). IEEE Computer Society Press, Washington, D.C., 2004, 199–206.
14. Ko, A.J., Myers, B.A., Coblenz, M., and Aung, H.H. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Transactions on Software Engineering* 33, 12 (Dec. 2006), 971–987.
15. Moaty, M., Faulring, A., Stylos, J., and Myers, B.A. Calcite: Completing code completion for constructors using crowds. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing* (Leganés-Madrid, Spain, Sept. 21–25). IEEE Computer Society Press, Washington, D.C., 2010, 15–22.
16. Nielsen, J. *Usability Engineering*. Academic Press, Boston, MA, 1993.
17. Oracle Corp. *Secure Coding Guidelines for the Java Programming Language, Version 4.0*, 2014; <http://www.oracle.com/technetwork/java/seccodeguide-139067.html>
18. Rama, G.M. and Kak, A. Some structural measures of API usability. *Software: Practice and Experience* 45, 1 (Jan. 2013), 75–110; https://engineering.purdue.edu/RVL/Publications/RamaKakAPIQ_SPE.pdf
19. Robillard, M. and DeLine, R. A field study of API learning obstacles. *Empirical Software Engineering* 16, 6 (Dec. 2011), 703–732.
20. Scheller, T. and Kuhn, E. Automated measurement of API usability: The API concepts framework. *Information and Software Technology* 61 (May 2015), 145–162.
21. Stylos, J., Busse, D.K., Graf, B., Ziegler, C., Ehret, R., and Karstens, J. A case study of API design for improved usability. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing* (Herrsching am Ammersee, Germany, Sept. 20–24). IEEE Computer Society Press, Washington, D.C., 2008, 189–192.
22. Stylos, J. and Clarke, S. Usability implications of requiring parameters in objects' constructors. In *Proceedings of the International Conference on Software Engineering* (Minneapolis, MN, May 20–26). IEEE Computer Society Press, Washington, D.C., 2007, 529–539.
23. Stylos, J., Faulring, A., Yang, Z., and Myers, B.A. Improving API documentation using API usage information. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing* (Corvallis, OR, Sept. 20–24). IEEE Computer Society Press, Washington, D.C., 2009, 119–126.
24. Stylos, J. and Myers, B.A. Mapping the space of API design decisions. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing* (Coeur d'Alene, ID, Sept. 23–27). IEEE Computer Society Press, Washington, D.C., 2007, 50–57.
25. Stylos, J. and Myers, B.A. The implications of method placement on API learnability. In *Proceedings of the 16th ACM SIGSOFT Symposium on Foundations of Software Engineering* (Atlanta, GA, Sept. 23–27). ACM Press, New York, 2008, 105–112.

Brad A. Myers (bam@cs.cmu.edu) is a professor in the Human-Computer Interaction Institute in the School of Computer Science at Carnegie Mellon University, Pittsburgh, PA.

Jeffrey Stylos (jstylos@us.ibm.com) is a software engineer at IBM in Littleton, MA, and received his Ph.D. in computer science at Carnegie Mellon University, Pittsburgh, PA, while doing research reported in this article.

Computers broadcast their secrets via inadvertent physical emanations that are easily measured and exploited.

BY DANIEL GENKIN, LEV PACHMANOV, ITAMAR PIPMAN, ADI SHAMIR, AND ERAN TROMER

Physical Key Extraction Attacks on PCs

CRYPTOGRAPHY IS UBIQUITOUS. Secure websites and financial, personal communication, corporate, and national secrets all depend on cryptographic algorithms operating correctly. Builders of cryptographic systems have learned (often the hard way) to devise algorithms and protocols with sound theoretical analysis, write software that implements them correctly, and robustly integrate them with the surrounding applications. Consequentially, direct attacks against state-of-the-art cryptographic software are getting increasingly difficult.

For attackers, ramming the gates of cryptography is not the only option. They can instead undermine the fortification by violating basic assumptions made by the cryptographic software. One such assumption is software can control its outputs. Our programming courses explain that programs produce their outputs through designated interfaces (whether print, write, send, or mmap); so, to keep a secret, the software just

needs to never output it or anything that may reveal it. (The operating system may be misused to allow someone else's process to peek into the program's memory or files, though we are getting better at avoiding such attacks, too.)

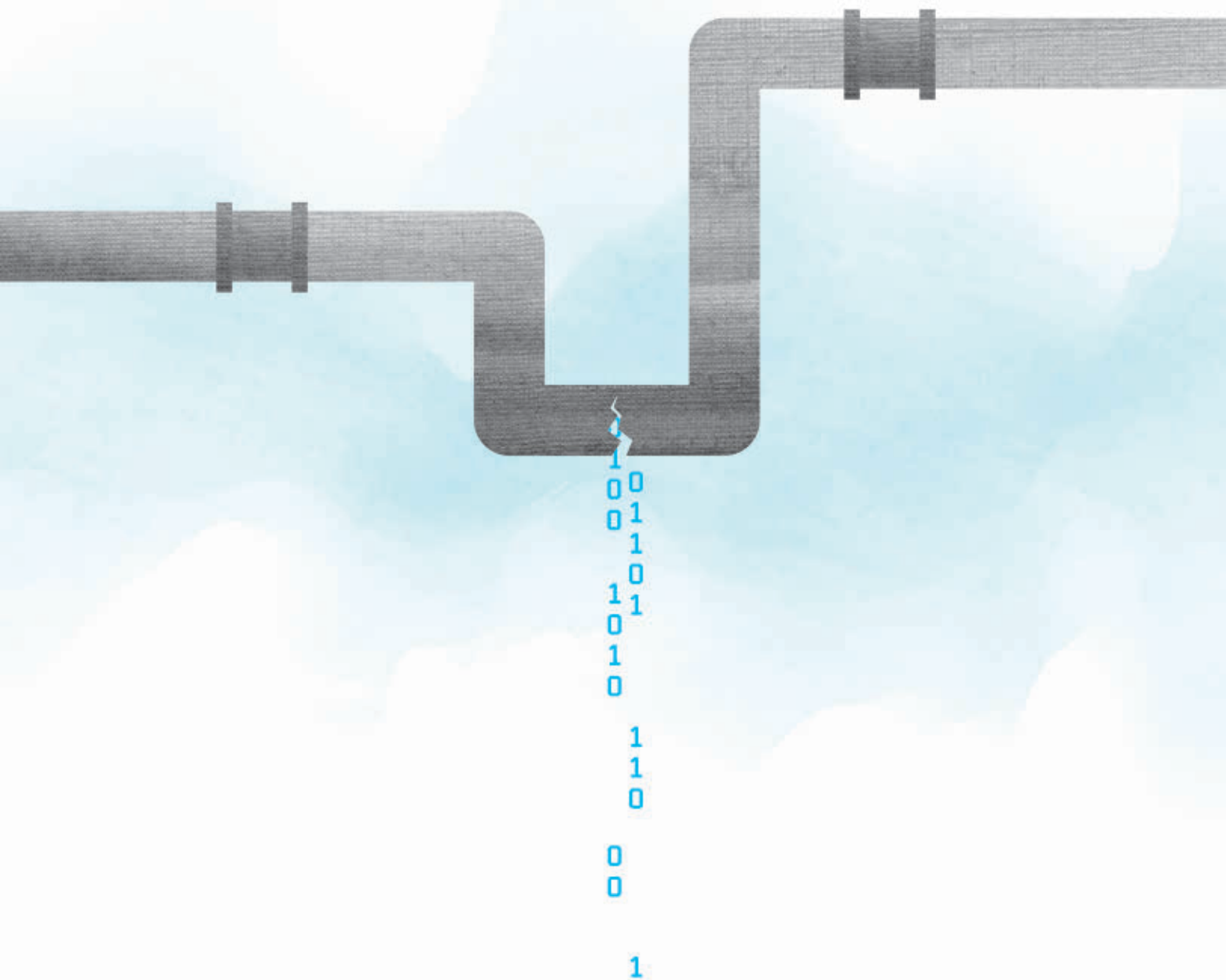
Yet programs' control over their own outputs is a convenient fiction, for a deeper reason. The hardware running the program is a physical object and, as such, interacts with its environment in complex ways, including electric currents, electromagnetic fields, sound, vibrations, and light emissions. All these "side channels" may depend on the computation performed, along with the secrets within it. "Side-channel attacks," which exploit such information leakage, have been used to break the security of numerous cryptographic implementations; see Anderson,² Kocher et al.,¹⁹ and Mangard et al.²³ and references therein.

Side channels on small devices.

Many past works addressed leakage from small devices (such as smart-cards, RFID tags, FPGAs, and simple embedded devices); for such devices, physical key extraction attacks have been demonstrated with devastating effectiveness and across multiple physical channels. For example, a device's power consumption is often correlated with the computation it is currently executing. Over the past two decades, this physical phenomenon has been used extensively for key extraction from small devices,^{19,23} often using powerful techniques, including differential power analysis.¹⁸

» key insights

- **Small differences in a program's data can cause large differences in acoustic, electric, and electromagnetic emanations as the program runs.**
- **These emanations can be measured through inexpensive equipment and used to extract secret data, even from fast and complex devices like laptop computers and mobile phones.**
- **Common hardware and software are vulnerable, and practical mitigation of these risks requires careful application-specific engineering and evaluation.**



The electromagnetic emanations from a device are likewise affected by the computation-correlated currents inside it. Starting with Agrawal et al.,¹ Gandolfi et al.,¹¹ and Quisquater and Samyde,²⁸ such attacks have been demonstrated on numerous small devices involving various cryptographic implementations.

Optical and thermal imaging of circuits provides layout information and coarse activity maps that are useful for reverse engineering. Miniature probes can be used to access individual internal wires in a chip, though such techniques require invasive disassembly of the chip package, as well as considerable technical expertise. Optical

emanations from transistors, as they switch state, are exploitable as a side channel for reading internal registers leading and extracting keys.²⁹

See Anderson² for an extensive survey of such attacks.

Vulnerability of PCs. Little was known, however, about the possibility of cryptographic attacks through physical side channels on modern commodity laptop, desktop, and server computers. Such “PC-class” computers (or “PCs,” as we call them here) are indeed very different from the aforementioned small devices, for several reasons.

First, a PC is a very complex environment—a CPU with perhaps one billion

transistors, on a motherboard with other circuitry and peripherals, running an operating system and handling various asynchronous events. All these introduce complexity, unpredictability, and noise into the physical emanations as the cryptographic code executes.

Second is speed. Typical side-channel techniques require the analog leakage signal be acquired at a bandwidth greater than the target’s clock rate. For PCs running GHz-scale CPUs, this means recording analog signals at multi-GHz bandwidths requiring expensive and delicate lab equipment, in addition to a lot of storage space and processing power.

Figure 1. An acoustic attack using a parabolic microphone (left) on a target laptop (right); keys can be extracted from a distance of 10 meters.



Figure 2. Measuring the chassis potential by touching a conductive part of the laptop; the wristband is connected to signal-acquisition equipment.



A third difference involves attack scenarios. Traditional techniques for side-channel attacks require long, uninterrupted physical access to the target device. Moreover, some such attacks involve destructive mechanical intrusion into the device (such as decapsulating chips). For small devices, these scenarios make sense; such devices are often easily stolen and sometimes even handed out to the attacker (such as in the form of cable TV subscription cards). However, when attacking other people's PCs, the attacker's physical access is often brief, constrained, and must proceed unobserved.

Note numerous side channels in PCs are known at the software level; timing,⁸ cache contention,^{6,26,27} and

many other effects can be used to glean sensitive information across the boundaries between processes or even virtual machines. Here, we focus on physical attacks that do not require deployment of malicious software on the target PC.

Our research thus focuses on two main questions: Can physical side-channel attacks be used to nonintrusively extract secret keys from PCs, despite their complexity and operating speed? And what is the cost of such attacks in time, equipment, expertise, and physical access?

Results. We have identified multiple side channels for mounting physical key-extraction attacks on PCs, applicable in various scenarios and offering

various trade-offs among attack range, speed, and equipment cost. The following sections explore our findings, as published in several recent articles.^{12,15,16}

Acoustic. The power consumption of a CPU and related chips changes drastically (by many Watts) depending on the computation being performed at each moment. Electronic components in a PC's internal power supply, struggling to provide constant voltage to the chips, are subject to mechanical forces due to fluctuations of voltages and currents. The resulting vibrations, as transmitted to the ambient air, create high-pitched acoustic noise, known as "coil whine," even though it often originates from capacitors. Because this noise is correlated with the ongoing computation, it leaks information about what applications are running and what data they process. Most dramatically, it can acoustically leak secret keys during cryptographic operations.

By recording such noise while a target is using the RSA algorithm to decrypt ciphertexts (sent to it by the attacker), the RSA secret key can be extracted within one hour for a high-grade 4,096-bit RSA key. We experimentally demonstrated this attack from as far as 10 meters away using a parabolic microphone (see Figure 1) or from 30cm away through a plain mobile phone placed next to the computer.

Electric. While PCs are typically grounded to the mains earth (through their power supply "brick," or grounded peripherals), these connections are, in practice, not ideal, so the electric potential of the laptop's chassis fluctuates. These fluctuations depend on internal currents, and thus on the ongoing computation. An attacker can measure the fluctuations directly through a plain wire connected to a conductive part of the laptop, or indirectly through any cable with a conductive shield attached to an I/O port on the laptop (such as USB, Ethernet, display, or audio). Perhaps most surprising, the chassis potential can be measured, with sufficient fidelity, even through a human body; human attackers need to touch only the target computer with a bare hand while their body potential is measured (see Figure 2).

This channel offers a higher bandwidth than the acoustic one, allowing

observation of the effect of individual key bits on the computation. RSA and ElGamal keys can thus be extracted from a signal obtained from just a few seconds of measurement, by touching a conductive part of the laptop's chassis, or by measuring the chassis potential from the far side of a 10-meter-long cable connected to the target's I/O port.

Electromagnetic. The computation performed by a PC also affects the electromagnetic field it radiates. By monitoring the computation-dependent electromagnetic fluctuations through an antenna for just a few seconds, it is possible to extract RSA and ElGamal secret keys. For this channel, the measurement setup is notably unintrusive and simple. A suitable electromagnetic probe antenna can be made from a simple loop of wire and recorded through an inexpensive software-defined radio USB dongle. Alternatively, an attacker can sometimes use a plain consumer-grade AM radio receiver, tuned close to the target's signal frequency, with its headphone output connected to a phone's audio jack for digital recording (see Figure 3).

Applicability. A surprising result of our research is how practical and easy are physical key-extraction side-channel attacks on PC-class devices, despite the devices' apparent complexity and high speed. Moreover, unlike previous attacks, our attacks require very little analog bandwidth, as low as 50kHz, even when attacking multi-GHz CPUs, thus allowing us to utilize new channels, as well as inexpensive and readily available hardware.

We have demonstrated the feasibility of our attacks using GnuPG (also known as GPG), a popular open source cryptographic software that implements both RSA and ElGamal. Our attacks are effective against various versions of GnuPG that use different implementations of the targeted cryptographic algorithm. We tested various laptop computers of different models from different manufacturers and running various operating systems, all “as is,” with no modification or case intrusions.

History. Physical side-channel attacks have been studied for decades in military and espionage contexts in the U.S. and NATO under the codename

TEMPEST. Most of this work remains classified. What little is declassified confirms the existence and risk of physical information leakage but says nothing about the feasibility of the key extraction scenarios discussed in this article. Acoustic leakage, in particular, has been used against electromechanical ciphers (Wright³¹ recounts how the British security agencies tapped a phone to eavesdrop on the rotors of a Hagelin electromechanical cipher machine); but there is strong evidence it was not recognized by the security services as effective against modern electronic computers.¹⁶

Non-Cryptographic Leakage

Peripheral devices attached to PCs are

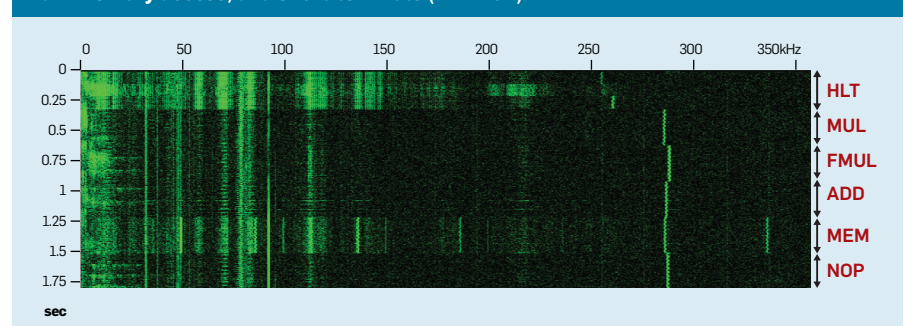
prone to side-channel leakage due to their physical nature and lower operating speed; for example, acoustic noise from keyboards can reveal keystrokes,³ printer-noise printed content,⁴ and status LEDs data on a communication line.²² Computer screens inadvertently broadcast their content as “van Eck” electromagnetic radiation that can be picked up from a distance;^{21,30} see Anderson² for a survey.

Some observations have also been made about physical leakage from PCs, though at a coarse level. The general activity level is easily gleaned from temperature,⁷ fan speed, and mechanical hard-disk movement. By tapping the computer's electric AC power, it is possible to identify the webpages

Figure 3. An electromagnetic attack using a consumer AM radio receiver placed near the target and recorded by a smartphone.



Figure 4. A spectrogram of an acoustic signal. The vertical axis is time (3.7 seconds), and the horizontal axis is frequency (0kHz–310kHz). Intensity represents instantaneous energy in the frequency band. The target is performing one-second loops of several x86 instructions: CPU sleep (HLT), integer multiplication (MUL), floating-point multiplication (FMUL), main memory access, and short-term idle (REP NOP).



loaded by the target's browser⁹ and even some malware.¹⁰ Tapping USB power lines makes it possible to identify when cryptographic applications are running.²⁵

The acoustic, electric, and electromagnetic channels can also be used to gather coarse information about a target's computations; Figure 4 shows a microphone recording of a PC, demonstrating loops of different operations have distinct acoustic signatures.

Cryptanalytic Approach

Coarse leakage is ubiquitous and easily demonstrated once the existence of the physical channel is recognized. However, there remains the question of whether the physical channels can be used to steal finer and more devastating information. The crown jewels, in this respect, are cryptographic keys, for three reasons. First, direct impact, as compromising cryptographic keys endangers all data and authorizations that depend on them. Second, difficulty, as cryptographic keys tend to be well protected and used in carefully crafted algorithms designed to resist attacks; so if even these keys can be extracted, it is a strong indication more pedestrian data can be also extracted. And third, commonality, as there is only a small number of popular cryptographic

algorithms and implementations, so compromising any of them has a direct effect on many deployed systems. Consequently, our research focused on key extraction from the most common public-key encryption schemes—RSA and ElGamal—as implemented by the popular GnuPG software.

When analyzing implementations of public-key cryptographic algorithms, an attacker faces the difficulties described earlier of complexity, noise, speed, and nonintrusiveness. Moreover, engineers implementing cryptographic algorithms try to make the sequence of executed operations very regular and similar for all secret keys. This is done to foil past attacks that exploit significant changes in control flow to deduce secrets, including timing attacks,⁸ cache contention attacks^{6,26,27} (such as a recent application to GnuPG^{32,33}), and many other types of attacks on small devices.

We now show how to overcome these difficulties, using a careful selection of the ciphertext to be decrypted by the algorithm. By combining the following two techniques for ciphertext selection, we obtain a key-dependent leakage that is robustly observable, even through low-bandwidth measurements.

Internal value poisoning. While the sequence of performed operations

is often decoupled from the secret key, the operands to these operations are often key-dependent. Moreover, operand values with atypical properties (such as operands containing many zero bits or that are unusually short) may trigger implementation-dependent corner cases. We thus craft special inputs (ciphertexts to be decrypted) that “poison” internal values occurring inside the cryptographic algorithm, so atypically structured operands occur at key-dependent times. Measuring leakage during such a poisoned execution can reveal at which operations these operands occurred, and thus leak key information.

Leakage self-amplification. In order to overcome a device's complexity and execution speed, an attacker can exploit the algorithm's own code to amplify its own leakage. By asking for decryption of a carefully chosen ciphertext, we create a minute change (compared to the decryption of a random-looking ciphertext) during execution of the innermost loop of the attacked algorithm. Since the code inside the innermost loop is executed many times throughout the algorithm, this yields an easily observable global change affecting the algorithm's entire execution.

GnuPG'S RSA Implementation

For concreteness in describing our basic attack method, we outline GnuPG's implementation of RSA decryption, as of version 1.4.14 from 2013. Later GnuPG versions revised their implementations to defend against the adaptive attack described here; we discuss these variants and corresponding attacks later in the article.

Notation. RSA key generation is done by choosing two large primes p , q , a public exponent e and a secret exponent d , such that $ed \equiv 1 \pmod{\Phi(n)}$ where $n = pq$ and $\Phi(n) = (p-1)(q-1)$. The public key is (n, e) and the private key is (p, q, d) . RSA encryption of a message m is done by computing $m^e \bmod n$, and RSA decryption of a ciphertext c is done by computing $c^d \bmod n$. GnuPG uses a common optimization for RSA decryption; instead of directly computing $m = c^d \bmod n$, it first computes $m_p = c^{d_p} \bmod p$, $m_q = c^{d_q} \bmod q$ (where d_p and d_q are derived from the secret key), then combines m_p and m_q

Algorithm 1. Modular exponentiation using square-and-always-multiply.

```

Input: Three integers  $c, d, q$  in binary representation such
        that  $d = d_1 \dots d_m$ .
Output:  $a = c^d \bmod q$ .
1: procedure MOD_EXP( $c, d, q$ )
2:    $c \leftarrow c \bmod q$ 
3:    $a \leftarrow 1$ 
4:   for  $i \leftarrow 1$  to  $m$  do
5:      $a \leftarrow a^2$ 
6:      $t \leftarrow a \cdot c$ 
7:     if  $d_i = 1$  then
8:        $a \leftarrow t$ 
9:   return  $a$ 

```

Algorithm 2. GnuPG's basic multiplication code.

```

Input: Two integers  $a = a_s \dots a_1$  and  $b = b_t \dots b_1$  of size  $s$ .
        and  $t$  limbs respectively
Output:  $a \cdot b$ .
1: procedure MUL_BASECASE( $a, b$ )
2:    $p \leftarrow a \cdot b_1$ 
3:   for  $i \leftarrow 2$  to  $t$  do
4:     if  $b_i \neq 0$  then  $\triangleright$  (and if  $b_i = 0$  do nothing)
5:        $p \leftarrow p + a \cdot b_i \cdot 2^{32 \cdot (i-1)}$ 
6:   return  $p$ 

```

into m using the Chinese Remainder Theorem. To fully recover the secret key, it suffices to learn any of its components (p , q , d , dp , or d_q); the rest can be deduced.

Square-and-always-multiply exponentiation. Algorithm 1 is pseudocode of the square-and-always-multiply exponentiation used by GnuPG 1.4.14 to compute m_p and m_q . As a countermeasure to the attack of Yarom and Falkner,³² the sequence of squarings and multiplications performed by Algorithm 1 is independent of the secret key. Note the modular reduction in line 2 and the multiplication in line 6. Both these lines are used by our attack on RSA—line 2 for poisoning internal values and line 6 for leakage self-amplification.

Since our attack uses GnuPG's multiplication routine for leakage self-amplification, we now analyze the code of GnuPG's multiplication routines.

Multiplication. For multiplying large integers (line 6), GnuPG uses a variant of the Karatsuba multiplication algorithm. It computes the product of two k -numbers a and b recursively, using the identity $ab = (2^{2k} + 2^k)a_Hb_H + 2^k(a_H - a_L)(b_L - b_L) + (2^k + 1)a_Lb_L$, where a_H , b_H are the most significant halves of a and b , respectively, and, similarly, a_L , b_L are the least significant halves of a and b .

The recursion's base case is a simple grade-school “long multiplication” algorithm, shown (in simplified form) in Algorithm 2. GnuPG stores large integers in arrays of 32-bit words, called limbs. Note how Algorithm 2 handles the case of zero limbs of b . Whenever a zero limb of b is encountered, the operation in line 5 is not executed, and the loop in line 3 proceeds to handle the next limb of b . This optimization is exploited by the leakage self-amplification component of our attack. Specifically, each of our chosen ciphertexts will cause a targeted bit of q to affect the number of zero limbs of b given to Algorithm 2 and thus the control flow in line 4 and thereby the side-channel leakage.

Adaptive Chosen Ciphertext Attack

We now describe our first attack on RSA, extracting the bits of the secret prime q , one by one. For each bit of q , denoted q_i , the attack chooses a ciphertext $c^{(i)}$ such that when $c^{(i)}$ is decrypted

by the target the side-channel leakage reveals the value of q_i . Eventually the entire q is revealed. The choice of each ciphertext depends on the key bits learned thus far, making it an adaptive chosen ciphertext attack.

This attack requires the target to decrypt ciphertexts chosen by the attacker, which is realistic since GnuPG is invoked by numerous applications to decrypt ciphertexts arriving via email messages, files, webpages, and chat messages. For example, Enigmail and GpgOL are popular plugins that add PGP/MIME encrypted-email capabilities to Mozilla Thunderbird and Outlook, respectively. They decrypt incoming email messages by passing them to GnuPG. If the target uses them, an attacker can remotely inject a chosen ciphertext into GnuPG by encoding the ciphertext as a PGP/MIME email (following RFC 3156) and sending it to the target.

Cryptanalysis. We can now describe the adaptive chosen ciphertext attack on GnuPG's RSA implementation.

Internal value poisoning. We begin by choosing appropriate ciphertexts that will poison some of the internal values inside Algorithm 1. Let p , q be two random k -bit primes comprising an RSA secret key; in the case of high-security 4,096-bit RSA, $k = 2,048$. GnuPG always generates RSA keys such that the most significant bit of p and q is set, thus $q_i = 1$. Assume we have already recovered the topmost $i - 1$ bits of q and define the ciphertext $c^{(i)}$ to be the k -bit ciphertext whose topmost $i - 1$ bits are the same as q , its i -th bit is 0 and whose remaining bits are set to 1. Consider the effects of decrypting $c^{(i)}$ on the intermediate values of Algorithm 1, depending on the secret key bit q_i .

Suppose $q_i = 1$. Then $c^{(i)} \leq q$, and this $c^{(i)}$ is passed as the argument c to Algorithm 1, where the modular reduction in line 2 returns $c = c^{(i)}$ (since $c^{(i)} \leq q$), so the lowest $k - i$ bits of c remain 1. Conversely, if $q_i = 0$, then $c^{(i)} > q$, so when $c^{(i)}$ is passed to Algorithm 1, the modular reduction in line 2 modifies the value of c . Since $c^{(i)}$ agrees with q on its topmost $i - 1$ bits, it holds that $q < c^{(i)} < 2q$, so in this case the modular reduction computes $c \leftarrow c - q$, which is a random-looking number of length $k - i$ bits.

We have thus obtained a connection between the i -th bit of q and the resulting structure of c after the modular reduction—either long and repetitive or short and random looking—thereby poisoning internal values in Algorithm 1.

Leakage self-amplification. To learn the i -th bit of q , we need to amplify the leakage resulting from this connection so it becomes physically distinguishable. Note the value c is used during the main loop of Algorithm 1 in line 6. Moreover, since the multiplication in line 6 is executed once per bit of d , we obtain that Algorithm 1 performs k multiplications by c , whose structure depends on q_i . We now analyze the effects of repetitive vs. random-looking second operand on the multiplication routine of GnuPG.

Suppose $c^{(i)} = 1$. Then c has its lowest $k - i$ bits set to 1. Next, c is passed to the Karatsuba-based multiplication routine as the second operand b . The result of $(b_L - b_H)$, as computed in the Karatsuba-based multiplication, will thus contain many zero limbs. This invariant, of having the second operand containing many zero limbs, is preserved by the Karatsuba-based multiplication all the way until the recursion reaches the base-case multiplication routine (Algorithm 2), where it affects the control flow in line 4, forcing the loop in line 3 to perform almost no multiplications.

Conversely, if $q_i = 0$, then c is random-looking, containing few (if any) zero limbs. When the Karatsuba-based multiplication routine gets c as its second operand b , the derived values stay random-looking throughout the recursion until the base case, where these random-looking values affect the control flow in line 4 inside the main loop of Algorithm 2, making it almost always perform a multiplication.

Our attack thus creates a situation where, during the entire decryption operation, the branch in line 4 of Algorithm 2 is either always taken or is never taken, depending on the current bit of q . During the decryption process, the branch in line 4 is evaluated numerous times (approximately 129,000 times for 4,096-bit RSA). This yields the desired self-amplification effect. Once q_i is extracted, we can compute the next chosen ciphertext c^{i+1} and proceed to ex-

Figure 5. Measuring acoustic leakage: (a) is the attacked target; (b) is a microphone picking up the acoustic emanations; (c) is the microphone power supply and amplifier; (d) is the digitizer; and the acquired signal is processed and displayed by the attacker's laptop (e).

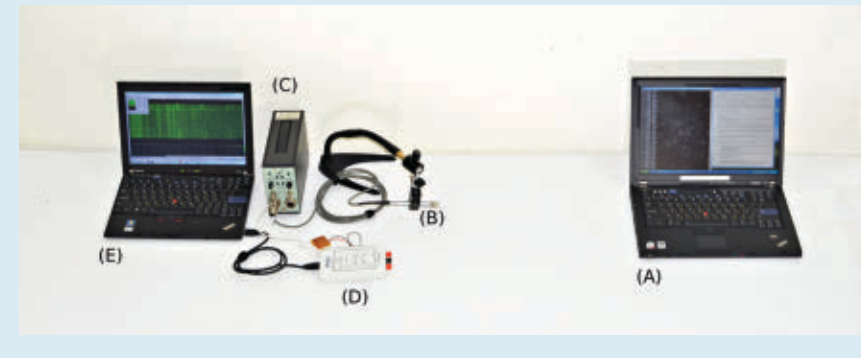
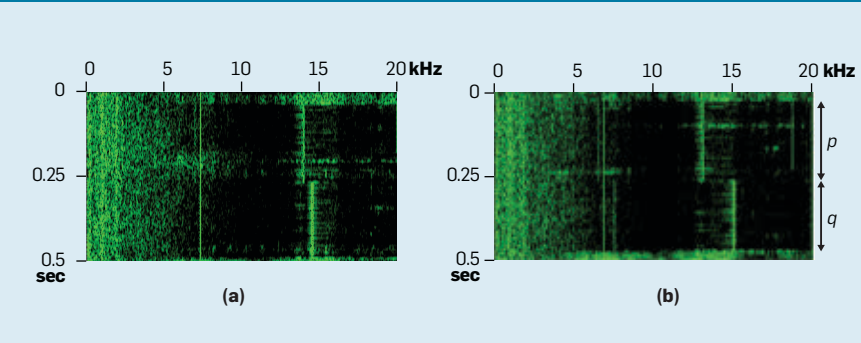


Figure 6. Acoustic emanations (0kHz–20kHz, 0.5 seconds) of RSA decryption during an adaptive chosen-ciphertext attack.



tract the next secret bit— q_{i+1} —through the same method.

The full attack requires additional components (such as error detection and recovery¹⁶).

Acoustic cryptanalysis of RSA. The basic experimental setup for measuring acoustic leakage consists of a microphone for converting mechanical air vibrations to electronic signals, an amplifier for amplifying the microphone's signals, a digitizer for converting the analog signal to a digital form, and software to perform signal processing and cryptanalytic deduction. Figure 1 and Figure 5 show examples of such setups using sensitive ultrasound microphones. In some cases, it even suffices to record the target through the built-in microphone of a mobile phone placed in proximity to the target and running the attacker's mobile app.¹⁶

Figure 6 shows the results of applying the acoustic attack for different values (0 or 1) of the attacked bit of q . Several effects are discernible. First, the transition between the two modular exponentiations (using the modulus p and q) is clearly visible. Second, note the acoustic signatures

of the second exponentiation is different between Figure 6a and Figure 6b. This is exactly the effect created by our attack, which can be utilized to extract the bits of q .

By applying the iterative attack algorithm described earlier, attacking each key bit at a time by sending the chosen ciphertext for decryption and learning the key bit from the measured acoustic signal, the attacker can fully extract the secret key. For 4,096-bit RSA keys (which, according to NIST recommendations, should remain secure for decades), key extraction takes approximately one hour.

Parallel load. This attack assumes decryption is triggered on an otherwise-idle target machine. If additional software is running concurrently, then the signal will be affected, but the attack may still be feasible. In particular, if other software is executed through timeslicing, then the irrelevant timeslices can be identified and discarded. If other, sufficiently homogenous software is executed on a different core, then (empirically) the signal of interest is merely shifted. Characterizing the general case is an

open question, but we conjecture that exploitable correlations will persist.

Non-Adaptive Chosen Ciphertext Attacks

The attack described thus far requires decryption of a new adaptively chosen ciphertext for every bit of the secret key, forcing the attacker to interact with the target computer for a long time (approximately one hour). To reduce the attack time, we turn to the electrical and electromagnetic channels, which offer greater analog bandwidth, though still orders of magnitude less than the target's CPU frequency. This increase in bandwidth allows the attacker to observe finer details about the operations performed by the target algorithm, thus requiring less leakage amplification.

Utilizing the increased bandwidth, our next attack trades away some of the leakage amplification in favor of reducing the number of ciphertexts. This reduction shortens the key-extraction time to seconds and, moreover, makes the attack non-adaptive, meaning the chosen ciphertexts can be sent to the target all at once (such as on a CD with a few encrypted files).

Cryptanalysis. The non-adaptive chosen ciphertext attack against square-and-always-multiply exponentiation (Algorithm 1) follows the approach of Yen et al.,³⁴ extracting the bits of d instead of q .

Internal value poisoning. Consider the RSA decryption of $c = n - 1$. As in the previous acoustic attack, c is passed to Algorithm 1, except this time, after the modular reduction in line 2, it holds that $c \equiv -1 \pmod{q}$. We now examine the effect of c on the squaring operation performed during the main loop of Algorithm 1.

First note the value of a during the execution of Algorithm 1 is always either 1 or -1 modulo q . Next, since $-1^2 \equiv 1^2 \equiv 1 \pmod{q}$, we have that the value of a in line 6 is always 1 modulo q . We thus obtain the following connection between the secret key bit d_{i-1} and the value of a at the start of the i -th iteration of Algorithm 1's main loop.

Suppose $d_{i-1} = 0$, so the branch in line 7 is not taken, making the value of a at the start of the i -th iteration be $1 \pmod{q} = 1$. Since GnuPG's internal representation does not truncate

leading zeros, a contains many leading zero limbs that are then passed to the squaring routine during the i -th iteration. Conversely, if $d_{i-1} = 1$, then the branch in line 7 is taken, making the value of a at the start of the i -th iteration be -1 modulo q , represented as $p - 1$. Since q is a randomly generated prime, the value of a , and therefore the value sent to the squaring routine during the i -th iteration, is unlikely to contain any zero limbs.

We have thus poisoned some of the internal values of Algorithm 1, creating a connection between the bits of d and the intermediate values of a during the exponentiation.

Amplification. GnuPG's squaring routines are implemented in ways similar to the multiplication routines, including the optimizations for handling zero limbs, yielding leakage self-amplification, as in an adaptive attack.

Since each iteration of the exponentiation's main loop leaks one bit of the secret d , all the bits d can be extracted from (ideally) a single decryption of a single ciphertext. In practice, a few measurements are needed to cope with noise, as discussed here.

Windowed exponentiation. Many RSA implementations, including GnuPG version 1.4.16 and newer, use an exponentiation algorithm that is faster than Algorithm 1. In such an implementation, the exponent d is split into blocks of m bits (typically $m = 5$), either contiguous blocks (in "fixed window" or " m -ary" exponentiation)

or blocks separated by runs of zero bits (in "sliding-window" exponentiation). The main loop, instead of handling the exponent one bit at a time, handles a whole block at every iteration, by multiplying a by c^x , where x is the block's value. The values c^x are pre-computed and stored in a lookup table (for all m -bit values x).

An adaptation of these techniques also allows attacking windowed exponentiation.¹² In a nutshell, we focus on each possible m -bit value x , one at a time, and identify which blocks in the exponent d , that is, which iterations of the main loop, contain x . This is done by crafting a ciphertext c such that $c^x \bmod q$ contains many zero limbs. Leakage amplification and measurement then work similarly to the acoustic and electric attacks described earlier. Once we identify where each x occurred, we aggregate these locations to deduce the full key d .

Electric attacks. As discussed earlier, the electrical potential on the chassis of laptop computers often fluctuates (in reference to the mains earth ground) in a computation-dependent way. In addition to measuring this potential directly using a plain wire connected to the laptop chassis, it is possible to measure the chassis potential from afar using the conductive shielding of any cable attached to one of the laptop's I/O ports (see Figure 7) or from nearby by touching an exposed metal part of the laptop's chassis, as in Figure 2.

To cope with noise, we measured the electric potential during a few (typically 10) decryption operations. Each recording was filtered and demodulated. We used frequency-demodulation since it produced best results compared to amplitude and phase demodulations. We then combined the recordings using correlation-based averaging, yielding a combined signal (see Figure 8). The successive bits of d can be deduced from this combined signal. Full key extraction, using non-adaptive electric measurements, requires only a few seconds of measurements, as opposed to an hour using the adaptive attack. We obtained similar results for ElGamal encryption; Genkin et al.¹⁵ offer a complete discussion.

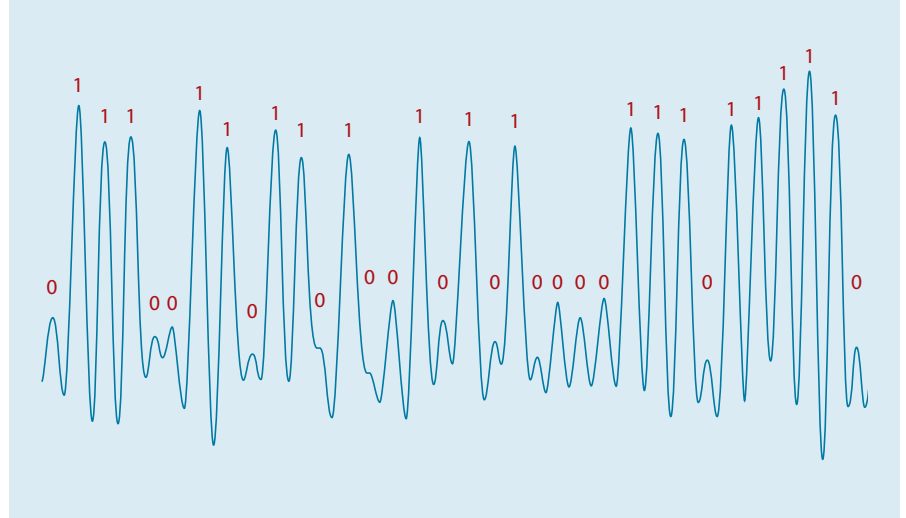
Electromagnetic attacks. The electromagnetic channel, which exploits computation-dependent fluctuations in the electromagnetic field surrounding the target, can also be used for key extraction. While this channel was previously used for attacks on small devices at very close proximity,^{1,11,28} the PC class of devices was only recently considered by Zajic and Prulovic³⁵ (without cryptographic applications).

Measuring the target's electromagnetic emanations requires an antenna, electronics for filtering and amplification, analog-to-digital conversion, and software for signal processing and cryptanalytic deduction. Prior works (on small devices) typically used cumbersome and expensive lab-grade

Figure 7. Measuring the chassis potential from the far side of an Ethernet cable (blue) plugged into the target laptop (10 meters away) through an alligator clip leading to measurement equipment (green wire).



Figure 8. A signal segment from an electric attack, after demodulating and combining measurements of several decryptions. Note the correlation between the signal (blue) and the correct key bits (red).



equipment. In our attacks,¹² we used highly integrated solutions that are small and inexpensive (such as a software-defined radio dongle, as in Figure 9, or a consumer-grade radio receiver recorded by a smartphone, as in Figure 3). Demonstrating how an untethered probe may be constructed from readily available electronics, we also built the Portable Instrument for Trace Acquisition (PITA), which is compact enough to be concealed, as in pita bread (see Figure 10).

Experimental results. Attacking RSA and ElGamal (in both square-and-always-multiply and windowed implementations) over the electromagnetic channel (sampling at 200 kSample/sec around a center frequency of 1.7MHz),

using the non-adaptive attack described earlier, we have extracted secret keys in a few seconds from a distance of half a meter.

Attacking other schemes and other devices. So far, we have discussed attacks on the RSA and ElGamal cryptosystems based on exponentiation in large prime fields. Similar attacks also target elliptic-curve cryptography. For example, we demonstrated key extraction from GnuPG's implementation of the Elliptic-Curve Diffie-Hellman scheme running on a PC;¹³ the attacker, in this case, can measure the target's electromagnetic leakage from an adjacent room through a wall.

Turning to mobile phones and tab-

lets, as well as to other cryptographic libraries (such as OpenSSL and iOS CommonCrypto), electromagnetic key extraction from implementations of the Elliptic Curve Digital Signature Algorithm has also been demonstrated, including attacks that are non-invasive,¹⁷ low-bandwidth,^{5,24} or both.¹⁴

Conclusion

Extraction of secret cryptographic keys from PCs using physical side channels is feasible, despite their complexity and execution speed. We have demonstrated such attacks on many public-key encryption schemes and digital-signature schemes, as implemented by popular cryptographic libraries, using inexpensive and readily available equipment, by various attack vectors and in multiple scenarios.

Hardware countermeasures.

Side-channel leakage can be attenuated through such physical means as sound-absorbing enclosures against acoustic attacks, Faraday cages against electromagnetic attacks, insulating enclosures against chassis and touch attacks, and photoelectric decoupling or fiber-optic connections against “far end of cable” attacks. However, these countermeasures are expensive and cumbersome. Devising inexpensive physical leakage protection for consumer-grade PCs is an open problem.

Software countermeasures.

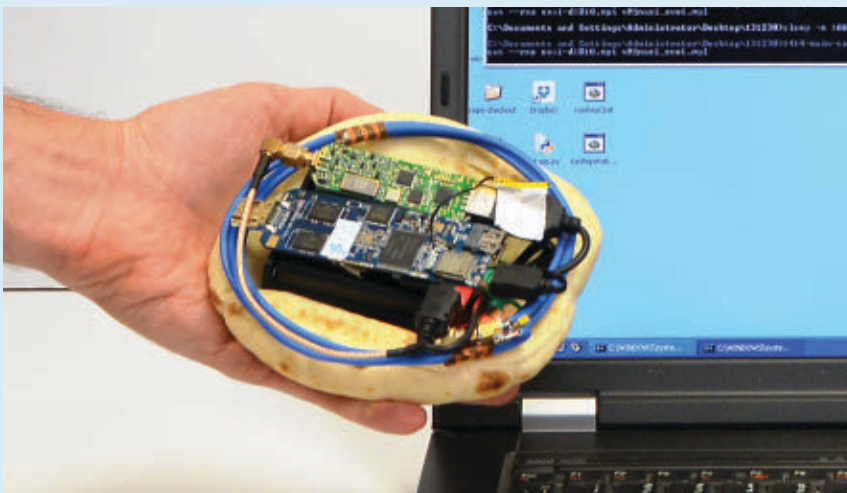
Given a characterization of a side channel, algorithms and their software implementations may be designed so the leakage through the given channel will not convey useful information. One such approach is “blinding,” or ensuring long operations (such as modular exponentiation) that involve sensitive values are, instead, performed on random dummy values and later corrected using an operation that includes the sensitive value but is much shorter and thus more difficult to measure (such as modular multiplication). A popular example of this approach is ciphertext randomization,²⁰ which was added to GnuPG following our observations and indeed prevents both the internal value poisoning and the leakage self-amplification components of our attacks.

However, such countermeasures require careful design and adaptation

Figure 9. Measuring electromagnetic emanations from a target laptop (left) through a loop of coax cable (handheld) recorded by a software-defined radio (right).



Figure 10. Extracting keys by measuring a laptop's electromagnetic emanations through a PITA device.



for every cryptographic scheme and leakage channel; moreover, they often involve significant cost in performance. There are emerging generic protection methods at the algorithmic level, using fully homomorphic encryption and cryptographic leakage resilience; however, their overhead is currently so great as to render them impractical.

Future work. To fully understand the ramifications and potential of physical side-channel attacks on PCs and other fast and complex devices, many questions remain open. What other implementations are vulnerable, and what other algorithms tend to have vulnerable implementations? In particular, can symmetric encryption algorithms (which are faster and more regular) be attacked? What other physical channels exist, and what signal processing and cryptanalytic techniques can exploit them? Can the attacks' range be extended (such as in acoustic attacks via laser vibrometers)? What level of threat do such channels pose in various real-world scenarios? Ongoing research indicates the risk extends well beyond the particular algorithms, software, and platforms we have covered here.

On the defensive side, we also raise three complementary questions: How can we formally model the feasible side-channel attacks on PCs? What engineering methods will ensure devices comply with the model? And what algorithms, when running on compliant devices, will provably protect their secrets, even in the presence of side-channel attacks?

Acknowledgments

This article is based on our previous research,^{12,13,15,16} which was supported by the Check Point Institute for Information Security, the European Union's 10th Framework Programme (FP10/2010-2016) under grant agreement no. 259426 ERC-CaC, a Google Faculty Research Award, the Leona M. & Harry B. Helmsley Charitable Trust, the Israeli Ministry of Science, Technology and Space, the Israeli Centers of Research Excellence I-CORE program (center 4/11), NATO's Public Diplomacy Division in the Framework of "Science for Peace," and the Simons Foundation and DIMACS/Simons Col-

laboration in Cryptography through National Science Foundation grant #CNS-1523467. C

References

- Agrawal, D., Archambeault, B., Rao, J.R., and Rohatgi, P. The EM side-channel(s). In *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems (CHES 2002)*. Springer, 2002, 29–45.
- Anderson, R.J. *Security Engineering: A Guide to Building Dependable Distributed Systems, Second Edition*. Wiley, 2008.
- Asonov, D. and Agrawal, R. Keyboard acoustic emanations. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 2004, 3–11.
- Backes, M., Dürmuth, M., Gerling, S., Pinkal, M., and Sporleder, C. Acoustic side-channel attacks on printers. In *Proceedings of the USENIX Security Symposium 2010*. USENIX Association, 2010, 307–322.
- Belgaric, P., Fouque, P.-A., Macario-Rat, G., and Tibouchi, M. Side-channel analysis of Weierstrass and Koblitz curve ECDSA on Android smartphones. In *Proceedings of the Cryptographers' Track of the RSA Conference (CT-RSA 2016)*. Springer, 2016, 236–252.
- Bernstein, D.J. Cache-timing attacks on AES. 2005; <http://cr.yp.to/papers.html#cachetiming>
- Bouchier, J., Dabbous, N., Kean, T., Marsh, C., and Naccache, D. *Thermocommunication*. Cryptology ePrint Archive, Report 2009/002, 2009; <https://eprint.iacr.org/2009/002>
- Brumley, D. and Boneh, D. Remote timing attacks are practical. *Computer Networks* 48, 5 (Aug. 2005), 701–716.
- Clark, S.S., Mustafa, H.A., Ransford, B., Sorber, J., Fu, K., and Xu, W. Current events: Identifying webpages by tapping the electrical outlet. In *Proceedings of the 18th European Symposium on Research in Computer Security (ESORICS 2013)*. Springer, Berlin, Heidelberg, 2013, 700–717.
- Clark, S.S., Ransford, B., Rahmati, A., Guineau, S., Sorber, J., Xu, W., and Fu, K. WattsUpDoc: Power side channels to nonintrusively discover untargeted malware on embedded medical devices. In *Proceedings of the USENIX Workshop on Health Information Technologies (HealthTech 2013)*. USENIX Association, 2013.
- Gandolfi, K., Mourtel, C., and Olivier, F. Electromagnetic analysis: Concrete results. In *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001)*. Springer, Berlin, Heidelberg, 2001, 251–261.
- Genkin, D., Pachmanov, L., Pipman, I., and Tromer, E. Stealing keys from PCs using a radio: Cheap electromagnetic attacks on windowed exponentiation. In *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems (CHES 2015)*. Springer, 2015, 207–228.
- Genkin, D., Pachmanov, L., Pipman, I., and Tromer, E. ECDH key-extraction via low-bandwidth electromagnetic attacks on PCs. In *Proceedings of the Cryptographers' Track of the RSA Conference (CT-RSA 2016)*. Springer, 2016, 219–235.
- Genkin, D., Pachmanov, L., Pipman, I., Tromer, E., and Yarom, Y. ECDSA Key Extraction from Mobile Devices via Nonintrusive Physical Side Channels. *Cryptology ePrint Archive*, Report 2016/230, 2016; <http://eprint.iacr.org/2016/230>
- Genkin, E., Pipman, I., and Tromer, E. Get your hands off my laptop: Physical side-channel key-extraction attacks on PCs. In *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems (CHES 2014)*. Springer, 2014, 242–260.
- Genkin, D., Shamir, A., and Tromer, E. RSA key extraction via low-bandwidth acoustic cryptanalysis. In *Proceedings of the Annual Cryptology Conference (CRYPTO 2014)*. Springer, 2014, 444–461.
- Kenworthy, G. and Rohatgi, P. Mobile device security: The case for side-channel resistance. In *Proceedings of the Mobile Security Technologies Conference (MoST)*, 2012; <http://mostconf.org/2012/papers/21.pdf>
- Kocher, P., Jaffe, J., and Jun, B. Differential power analysis. In *Proceedings of the Annual Cryptology Conference (CRYPTO 1999)*. Springer, 1999, 388–397.
- Kocher, P., Jaffe, J., Jun, B., and Rohatgi, P. *Introduction to differential power analysis*. *Journal of Cryptographic Engineering* 1, 1 (2011), 5–27.
- Kocher, P.C. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Proceedings of the Annual Cryptology Conference (CRYPTO 1996)*. Springer, 1996, 104–113.
- Kuhn, M.G. *Compromising Emanations: Eavesdropping Risks of Computer Displays*. Ph.D. Thesis and Technical Report UCAM-CL-TR-577. University of Cambridge Computer Laboratory, Cambridge, U.K., Dec. 2003; <https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-577.pdf>
- Loughry, J. and Umphress, D.A. Information leakage from optical emanations. *ACM Transactions on Information Systems Security* 5, 3 (Aug. 2002), 262–289.
- Mangard, S., Oswald, E., and Popp, T. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, Berlin, Heidelberg, 2007.
- Nakano, Y., Souissi, Y., Nguyen, R., Sauvage, L., Danger, J., Guille, S., Kiyomoto, S., and Miyake, Y. A pre-processing composition for secret key recovery on Android smartphones. In *Proceedings of the International Workshop on Information Security Theory and Practice (WISTP 2014)*. Springer, Berlin, Heidelberg, 2014.
- Oren, Y. and Shamir, A. How not to protect PCs from power analysis. Presented at the Annual Cryptology Conference (CRYPTO 2006) rump session, 2006; <http://iss.ojne.ro/HowNotToProtectPCsFromPowerAnalysis>
- Osvik, D.A., Shamir, A., and Tromer, E. Cache attacks and countermeasures: The case of AES. In *Proceedings of the Cryptographers' Track of the RSA Conference (CT-RSA 2006)*. Springer, 2006, 1–20.
- Percival, C. Cache missing for fun and profit. In *Proceedings of the BSDCan Conference*, 2005; <http://www.daemonology.net/hyperthreading-considered-harmful>
- Quisquater, J.-J. and Samyde, D. Electromagnetic analysis (EMA): Measures and countermeasures for smartcards. In *Proceedings of the Smart Card Programming and Security: International Conference on Research in Smart Cards (E-smart 2001)*. Springer, 2001, 200–210.
- Skorobogatov, S. *Optical Surveillance on Silicon Chips*. University of Cambridge, Cambridge, U.K., 2009; http://www.cl.cam.ac.uk/~sps32/SG_talk_OSSC_a.pdf
- van Eck, W. Electromagnetic radiation from video display units: An eavesdropping risk? *Computers and Security* 4, 4 (Dec. 1985), 269–286.
- Wright, P. *Spycatcher*. Viking Penguin, New York, 1987.
- Yarom, Y. and Falkner, K. FLUSH+RELOAD: A high-resolution, low-noise, L3 cache side-channel attack. In *Proceedings of the USENIX Security Symposium 2014*. USENIX Association, 2014, 719–732.
- Yarom, Y., Liu, F., Ge, Q., Heiser, G., and Lee, R.B. Last-level cache side-channel attacks are practical. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 2015, 606–622.
- Yen, S.-M., Lien, W.-C., Moon, S.-J., and Ha, J. Power analysis by exploiting chosen message and internal collisions: Vulnerability of checking mechanism for RSA decryption. In *Proceedings of the International Conference on Cryptology in Malaysia (MyCrypt 2005)*. Springer, 2005, 183–195.
- Zajic, A. and Prvulovic, M. Experimental demonstration of electromagnetic information leakage from modern processor-memory systems. *IEEE Transactions on Electromagnetic Compatibility* 56, 4 (Aug. 2014), 885–893.

Daniel Genkin (danielg3@cs.technion.ac.il) is a Ph.D. candidate in the Computer Science Department at Technion-Israel Institute of Technology, Haifa, Israel, and a research assistant in the Blavatnik School of Computer Science at Tel Aviv University, Israel.

Lev Pachmanov (levp@tau.ac.il) is a master's candidate in the Blavatnik School of Computer Science at Tel Aviv University, Israel.

Itamar Pipman (itamarpi@tau.ac.il) is a master's candidate in the Blavatnik School of Computer Science at Tel Aviv University, Israel.

Adi Shamir (adi.shamir@weizmann.ac.il) is a professor in the faculty of Mathematics and Computer Science at the Weizmann Institute of Science, Rehovot, Israel.

Eran Tromer (tromer@cs.tau.ac.il) is a senior lecturer in the Blavatnik School of Computer Science at Tel Aviv University, Israel.

Copyright held by authors.

Randomization offers new benefits for large-scale linear algebra computations.

BY PETROS DRINEAS AND MICHAEL W. MAHONEY

RandNLA: Randomized Numerical Linear Algebra

MATRICES ARE UBIQUITOUS in computer science, statistics, and applied mathematics. An $m \times n$ matrix can encode information about m objects (each described by n features), or the behavior of a discretized differential operator on a finite element mesh; an $n \times n$ positive-definite matrix can encode the correlations between all pairs of n objects, or the edge-connectivity between all pairs of nodes in a social network; and so on. Motivated largely by technological developments that generate extremely large scientific and Internet datasets, recent years have witnessed exciting developments in the theory and practice of matrix algorithms. Particularly remarkable is the use of *randomization*—typically assumed to be a property of the input data due to, for example, noise in the data

generation mechanisms—as an algorithmic or computational resource for the development of improved algorithms for fundamental matrix problems such as matrix multiplication, least-squares (LS) approximation, low-rank matrix approximation, and Laplacian-based linear equation solvers.

Randomized Numerical Linear Algebra (RandNLA) is an interdisciplinary research area that exploits randomization as a computational resource to develop improved algorithms for large-scale linear algebra problems.³² From a foundational perspective, RandNLA has its roots in theoretical computer science (TCS), with deep connections to mathematics (convex analysis, probability theory, metric embedding theory) and applied mathematics (scientific computing, signal processing, numerical linear algebra). From an applied perspective, RandNLA is a vital new tool for machine learning, statistics, and data analysis. Well-engineered implementations have already outperformed highly optimized software libraries for ubiquitous problems such as least-squares,^{4,35} with good scalability in parallel and distributed environments.⁵² Moreover, RandNLA promises a sound algorithmic and statistical foundation for modern large-scale data analysis.

» key insights

- Randomization isn't just used to model noise in data; it can be a powerful computational resource to develop algorithms with improved running times and stability properties as well as algorithms that are more interpretable in downstream data science applications.
- To achieve best results, random sampling of elements or columns/rows must be done carefully; but random projections can be used to transform or rotate the input data to a random basis where simple uniform random sampling of elements or rows/columns can be successfully applied.
- Random sketches can be used directly to get low-precision solutions to data science applications; or they can be used indirectly to construct preconditioners for traditional iterative numerical algorithms to get high-precision solutions in scientific computing applications.



An Historical Perspective

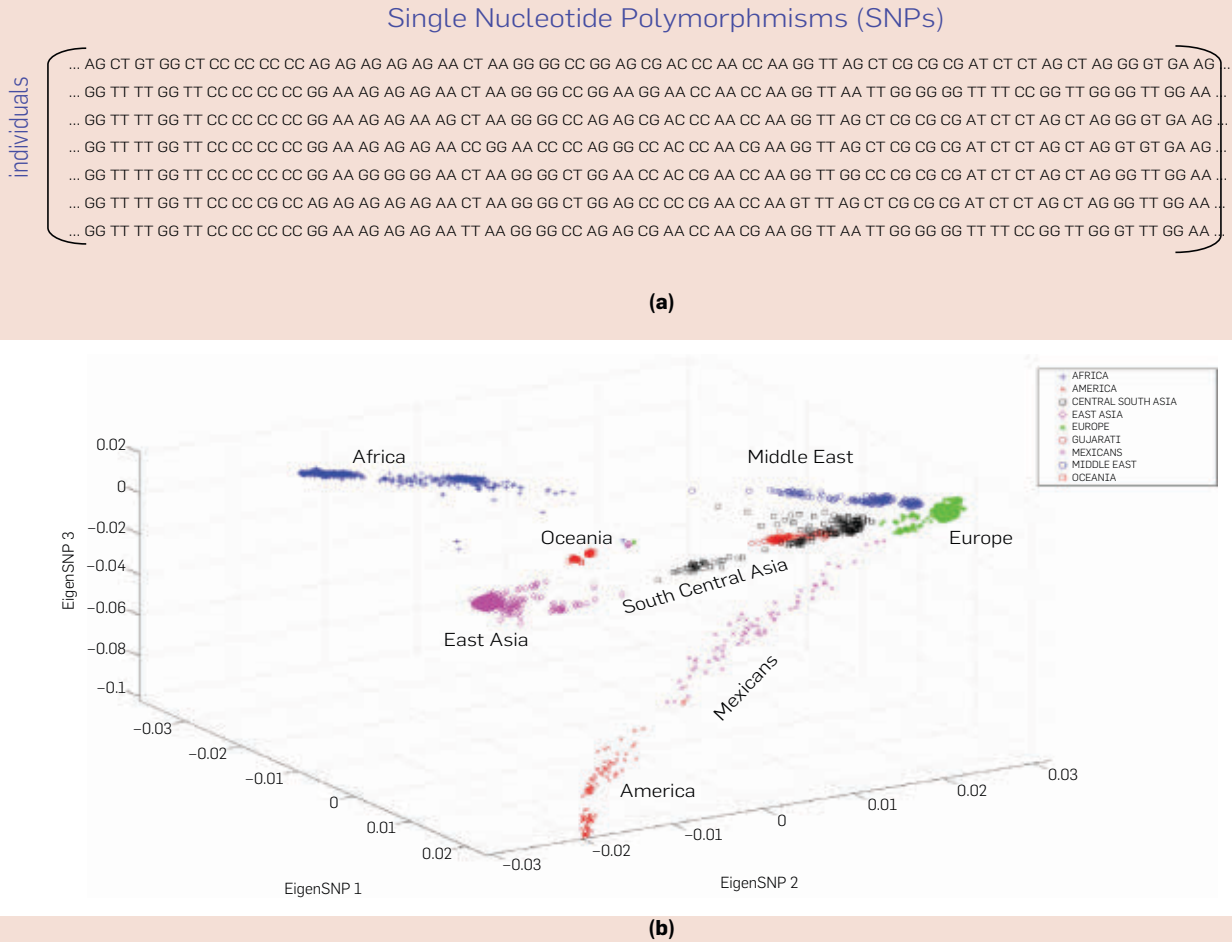
To get a broader sense of RandNLA, recall that linear algebra—the mathematics of vector spaces and linear mappings between vector spaces—has had a long history in large-scale (by the standards of the day) statistical data analysis.⁴⁶ For example, the least-squares method is due to Gauss, Legendre, and others, and was used in the early 1800s for fitting linear equations to data to determine planet orbits. Low-rank approximations based on Principal Component Analysis (PCA) are due to Pearson, Hotelling, and others, and were used in the early 1900s for exploratory data analysis and for making predictive models. Such methods are of interest for many reasons, but especially if there is noise or randomness *in the data*, because the leading principal components then tend to capture the signal and remove the noise.

With the advent of the digital computer in the 1950s, it became apparent that, even when applied to well-posed problems, many algorithms performed poorly in the presence of the finite precision that was used to represent real numbers. Thus, much of the early work in computer science focused on solving discrete approximations to continuous numerical problems. Work by Turing and von Neumann (then Householder, Wilkinson, and others) laid much of the foundations for scientific computing and NLA.^{48,49} Among other things, this led to the introduction of problem-specific complexity measures (for example, the condition number) that characterize the behavior of an input for a specific class of algorithms (for example, iterative algorithms).

A split then occurred in the nascent field of computer science. Continuous

linear algebra became the domain of applied mathematics, and much of computer science theory and practice became discrete and combinatorial.⁴⁴ Nearly all subsequent work in scientific computing and NLA has been deterministic (a notable exception being the work on integral evaluation using the Markov Chain Monte Carlo method). This led to high-quality codes in the 1980s and 1990s (LINPACK, EISPACK, LAPACK, ScaLAPACK) that remain widely used today. Meanwhile, Turing, Church, and others began the study of computation per se. It became clear that several seemingly different approaches (recursion theory, the λ -calculus, and Turing machines) defined the same class of functions; and this led to the belief in TCS that the concept of computability is formally captured in a

Figure 1. (a) Matrices are a common way to model data. In genetics, for example, matrices can describe data from tens of thousands of individuals typed at millions of Single Nucleotide Polymorphisms or SNPs (loci in the human genome). Here, the (i, j) th entry is the genotype of the i th individual at the j th SNP. (b) PCA/SVD can be used to project every individual on the top left singular vectors (or “eigenSNPs”), thereby providing a convenient visualization of the “out of Africa hypothesis” well known in population genetics.



qualitative and robust way by these three equivalent approaches, *independent of the input data*. Many of these developments were deterministic; but, motivated by early work on the Monte Carlo method, randomization—where the randomness is *inside the algorithm* and the algorithm is applied to arbitrary or worst-case data—was introduced and exploited as a powerful computational resource.

Recent years have seen these two very different perspectives start to converge. Motivated by modern massive dataset problems, there has been a great deal of interest in developing algorithms with improved running times and/or improved statistical properties that are more appropriate for obtaining insight from the enormous quantities of noisy data that is now being generated. At the center of

these developments is work on novel algorithms for linear algebra problems, and central to this is work on RandNLA algorithms.^a In this article, we will describe the basic ideas that underlie recent developments in this interdisciplinary area.

For a prototypical data analysis example where RandNLA methods have been applied, consider Figure 1, which illustrates an application in genetics³⁸ (although the same RandNLA methods have been applied in astronomy, mass spectrometry imaging, and related areas^{33,38,53,54}). While the low-dimensional PCA plot illustrates the famous correlation

^a Avron et al., in the first sentence of their Blendenpik paper, observe that RandNLA is “arguably the most exciting and innovative idea to have hit linear algebra in a long time.”⁴

between geography and genetics, there are several weaknesses of PCA/SVD-based methods. One is running time: computing PCA/SVD approximations of even moderately large data matrices is expensive, especially if it needs to be done many times as part of cross validation or exploratory data analysis. Another is interpretability: in general, eigenSNPs (that is, eigenvectors of individual-by-SNP matrices) as well as other eigenfeatures don’t “mean” anything in terms of the processes generating the data. Both issues have served as motivation to design RandNLA algorithms to compute PCA/SVD approximations faster than conventional numerical methods as well as to identify actual features (instead of eigenfeatures) that might be easier to interpret for domain scientists.

Basic RandNLA Principles

RandNLA algorithms involve taking an input matrix; constructing a “sketch” of that input matrix—where a sketch is a smaller or sparser matrix that represents the essential information in the original matrix—by random sampling; and then using that sketch as a surrogate for the full matrix to help compute quantities of interest. To be useful, the sketch should be similar to the original matrix in some way, for example, small residual error on the difference between the two matrices, or the two matrices should have similar action on sets of vectors or in downstream classification tasks. While these ideas have been developed in many ways, several basic design principles underlie much of RandNLA: (i) randomly sample, in a careful data-dependent manner, a small number of elements from an input matrix to create a much sparser sketch of the original matrix; (ii) randomly sample, in a careful data-dependent manner, a small number of columns and/or rows from an input matrix to create a much smaller sketch of the original matrix; and (iii) preprocess an input matrix with a random-projection-type matrix, in order to “spread out” or uniformize the information in the original matrix, and then use naïve data-independent uniform sampling of rows/columns/elements in order to create a sketch.

Element-wise sampling. A naïve way to view an $m \times n$ matrix A is an array of numbers: these are the mn elements of the matrix, and they are denoted by A_{ij} (for all $i = 1, \dots, m$ and all $j = 1, \dots, n$). It is therefore natural to consider the following approach in order to create a small sketch of a matrix A : instead of keeping all its elements, randomly sample and keep a small number of them. Algorithm 1 is a meta-algorithm that samples s elements from a matrix A in independent, identically distributed trials, where in each trial a single element of A is sampled with respect to the importance sampling probability distribution p_{ij} . The algorithm outputs a matrix $\tilde{A}$ that contains precisely the selected elements of A , after appropriate rescaling. This rescaling is fundamental from a statistical perspective: the sketch $\tilde{A}$ is an estimator for A . This rescaling makes it an unbiased estimator since, element-wise, the expectation of the

estimator matrix $\tilde{A}$ is equal to the original matrix A .

Algorithm 1 A meta-algorithm for element-wise sampling

Input: $m \times n$ matrix A ; integer $s > 0$ denoting the number of elements to be sampled; probability distribution p_{ij} ($i = 1, \dots, m$ and $j = 1, \dots, n$) with $\sum_{i,j} p_{ij} = 1$.

1. Let $\tilde{A}$ be an all-zeros $m \times n$ matrix.
2. For $t = 1$ to s ,

- **Randomly sample** one element of A using the probability distribution p_{ij} .
- Let $A_{i_t j_t}$ denote the sampled element and set

$$\tilde{A}_{i_t j_t} := \tilde{A}_{i_t j_t} + \frac{1}{s p_{i_t j_t}} A_{i_t j_t}. \quad (1)$$

Output: Return the $m \times n$ matrix $\tilde{A}$.

How to sample is, of course, very important. A simple choice is to perform uniform sampling, that is, set $p_{ij} = 1/mn$, for all i, j , and sample each element with equal probability. While simple, this suffers from obvious problems: for example, if all but one of the entries of the original matrix equal zero, and only a single non-zero entry exists, then the probability of sampling the single non-zero entry of A using uniform sampling is negligible. Thus, the estimator would have very large variance, in which case the sketch would, with high probability, fail to capture the relevant structure of the original matrix. Qualitatively improved results can be obtained by using nonuniform data-dependent importance sampling distributions. For example, sampling larger elements (in absolute value) with higher probability is advantageous in terms of variance reduction and can be used to obtain worst-case additive-error bounds for low-rank matrix approximation.^{1,2,18,28} More elaborate probability distributions (the so-called element-wise leverage scores that use information in the singular subspaces of A ¹⁰) have been shown to provide still finer results.

The first results for Algorithm 1² showed that if one chooses entries with probability proportional to their squared-magnitudes (that is, if $p_{ij} = A_{ij}^2 / \sum_{i,j} A_{ij}^2$, in which case larger magnitude entries are more likely to be chosen), then the sketch $\tilde{A}$ is similar

to the original matrix A , in the sense that the error matrix, $A - \tilde{A}$, has, with high probability, a small spectral norm. A more refined analysis¹⁸ showed that

$$\|A - \tilde{A}\|_2 \leq O\left(\sqrt{\frac{(m+n) \ln(m+n)}{s}}\right) \|A\|_F, \quad (2)$$

where $\|\cdot\|_2$ and $\|\cdot\|_F$ are the spectral and Frobenius norms, respectively, of the matrix.^b If the spectral norm of the difference $A - \tilde{A}$ is small, then $\tilde{A}$ can be used as proxy for A in applications. For example, one can use $\tilde{A}$ to approximate the spectrum (that is, the singular values and singular vectors) of the original matrix.² If s is set to be a constant multiple of $(m+n) \ln(m+n)$, then the error scales with the Frobenius norm of the matrix. This leads to an additive-error low-rank matrix approximation algorithm, in which $\|A\|_F$ is the scale of the additional additive error.² This is a large scaling factor, but improving upon this with element-wise sampling, even in special cases, is a challenging open problem.

The mathematical techniques used in the proof of these element-wise sampling results exploit the fact that the residual matrix $A - \tilde{A}$ is a random matrix whose entries have zero mean and bounded variance. Bounding the spectral norm of such matrices has a long history in random matrix theory.⁵⁰ Early RandNLA element-wise sampling bounds² used a result of Füredi and Komlós on the spectral norm of symmetric, zero mean matrices of bounded variance.²⁰ Subsequently, Drineas and Zouzias¹⁸ introduced the idea of using matrix measure concentration inequalities^{37,40,47} to simplify the proofs, and follow-up work¹⁸ has improved these bounds.

Row/column sampling. A more sophisticated way to view a matrix A is as a linear operator, in which case the role of rows and columns becomes more central. Much RandNLA research has focused on sketching a matrix by keeping only a few of its rows and/or

^b In words, the spectral norm of a matrix measures how much the matrix elongates or deforms the unit ball in the worst case, and the Frobenius norm measures how much the matrix elongates or deforms the unit ball on average. Sometimes the spectral norm may have better properties especially when dealing with noisy data, as discussed by Achlioptas and McSherry.²

columns. This method of sampling predates element-wise sampling algorithms,¹⁹ and it leads to much stronger worst-case bounds.^{15,16}

Algorithm 2 A meta-algorithm for row sampling

Input: $m \times n$ matrix A ; integer $s > 0$ denoting the number of rows to be sampled; probabilities p_i ($i = 1, \dots, m$) with $\sum_i p_i = 1$.

1. Let $\tilde{A}$ be the empty matrix.
2. **For** $t = 1$ **to** s ,
 - **Randomly sample** one row of A using the probability distribution p_i .
 - Let $A_{i_t^*}$ denote the sampled row and set

$$\tilde{A}_{i_t^*} := \frac{1}{\sqrt{s p_{i_t^*}}} A_{i_t^*}. \quad (3)$$

Output: Return the $s \times n$ matrix $\tilde{A}$.

Consider the meta-algorithm for row sampling (column sampling is analogous) presented in Algorithm 2. Much of the discussion of Algorithm 1 is relevant to Algorithm 2. In particular, Algorithm 2 samples s rows of A in independent, identically distributed trials according to the input probabilities p_i ; and the output matrix $\tilde{A}$ contains precisely the selected rows of A , after a rescaling that ensures un-biasedness of appropriate estimators (for example, the expectation of $\tilde{A}^T \tilde{A}$ is equal to $A^T A$, element-wise).^{13,19} In addition, uniform sampling can easily lead to very poor results, but qualitatively improved results can be obtained by using non-uniform, data-dependent, importance sampling distributions. Some things, however, are different: the dimension of the sketch $\tilde{A}$ is different than that of the original matrix A . The solution is to measure the quality of the sketch by comparing the difference between the matrices $A^T A$ and $\tilde{A}^T \tilde{A}$. The simplest nonuniform distribution is known as ℓ_2 sampling or *norm-squared sampling*, in which p_i is proportional to square of the Euclidean norm of the i th row^c:

$$p_i = \frac{\|A_{i^*}\|_2^2}{\sum_i \|A_{i^*}\|_2^2} = \frac{\|A_{i^*}\|_2^2}{\|A\|_F^2}. \quad (4)$$

^c We will use the notation A_{i^*} to denote the i th row of A as a row vector.

Motivated by modern massive dataset problems, there has been a great deal of interest in developing algorithms with improved running times and/or improved statistical properties that are more appropriate for obtaining insight from the enormous quantities of noisy data now being generated.

When using norm-squared sampling, one can prove that

$$\|A^T A - \tilde{A}^T \tilde{A}\|_F \leq \frac{1}{\sqrt{s}} \|A\|_F^2 \quad (5)$$

holds in expectation (and thus, by standard arguments, with high probability) for arbitrary A .^{13,19,d} The proof of Equation (5) is a simple exercise using basic properties of expectation and variance. This result can be generalized to approximate the product of two arbitrary matrices A and B .¹³ Proving such bounds with respect to other matrix norms is more challenging but very important for RandNLA. While Equation (5) trivially implies a bound for $\|A^T A - \tilde{A}^T \tilde{A}\|_2$, proving a better spectral norm error bound necessitates the use of more sophisticated methods such as the Khintchine inequality or matrix-Bernstein inequalities.^{42,47}

Bounds of the form of Equation (5) immediately imply that $\tilde{A}$ can be used as a proxy for A , for example, in order to approximate its (top few) singular values and singular vectors. Since $\tilde{A}$ is an $s \times n$ matrix, with $s \ll n$, computing its singular values and singular vectors is a very fast task that scales linearly with n . Due to the form of Equation (5), this leads to additive-error low-rank matrix approximation algorithms, in which $\|A\|_F$ is the scale of the additional additive error.¹⁹ That is, while norm-squared sampling avoids pitfalls of uniform sampling, it results in additive-error bounds that are only comparable to what element-wise sampling achieves.^{2,19}

To obtain stronger and more useful bounds, one needs information about the geometry or subspace structure of the high-dimensional Euclidean space spanned by the columns of A (if $m \gg n$) or the space spanned by the best rank- k approximation to A (if $m \sim n$). This can be achieved with *leverage score sampling*, in which p_i is proportional to

^d That is, a provably good approximation to the product $A^T A$ can be computed using just a few rows of A ; and these rows can be found by sampling randomly according to a simple data-dependent importance sampling distribution. This matrix multiplication algorithm can be implemented in one pass over the data from external storage, using only $O(sm)$ additional space and $O(s^2 n)$ additional time.

the i th leverage score of A . To define these scores, for simplicity assume that $m \gg n$ and that U is any $m \times n$ orthogonal matrix spanning the column space of A .^e In this case, $U^T U$ is equal to the identity and $U U^T = P_A$ is an m -dimensional projection matrix onto the span of A . Then, the importance sampling probabilities of Equation (4), applied to U , equal

$$p_i = \frac{\|U_{i*}\|_2^2}{\sum_i \|U_{i*}\|_2^2} = \frac{1}{n} (P_A)_{ii}. \quad (6)$$

Due to their historical importance in regression diagnostics and outlier detection, the p_i 's in Equation (6) are known as *statistical leverage scores*.^{9,14} In some applications of RandNLA, the largest leverage score is called the *coherence* of the matrix.^{8,14}

Importantly, while one can naïvely compute these scores via Equation (6) by spending $O(mn^2)$ time to compute U exactly, this is *not* necessary.¹⁴ Let Π be the fast Hadamard Transform as used in Drineas et al.¹⁴ or the input-sparsity-time random projection of Refs.^{12,34,36} Then, in $O(mn^2)$ time, one can compute the R matrix from a QR decomposition of ΠA and from that compute $1 \pm \epsilon$ relative-error approximations to all the leverage scores.¹⁴

In RandNLA, one is typically interested in proving that

$$\|U^T U - \tilde{U}^T \tilde{U}\|_2 = \|I - \tilde{U}^T \tilde{U}\|_2 \leq \epsilon, \quad (7)$$

either for arbitrary $\epsilon \in (0, 1)$ or for some fixed $\epsilon \in (0, 1)$. Approximate matrix multiplication bounds of the form of Equation (7) are very important in RandNLA algorithm design since the resulting sketch $\tilde{A}$ preserves rank properties of the original data matrix A and provides a *subspace embedding*: from the NLA perspective, this is simply an acute perturbation from the original high-dimensional space to a much lower dimensional space.²² From the TCS perspective, this provides bounds analogous to the usual Johnson–Lindenstrauss bounds, except that it preserves the geometry of the entire subspace.⁴³

^e A generalization holds if $m \sim n$: in this case, U is any $m \times k$ orthogonal matrix spanning the best rank- k approximation to the column space of A , and one uses the *leverage scores relative to the best rank- k approximation to A* .^{14,16,33}

Subspace embeddings were first used in RandNLA in a data-aware manner (meaning, by looking at the input data to compute exact or approximate leverage scores¹⁴) to obtain sampling-based relative-error approximation to the LS regression and related low-rank CX/CUR approximation problems.^{15,16} They were then used in a data-oblivious manner (meaning, in conjunction with a random projection as a preconditioner) to obtain projection-based relative-error approximation to several RandNLA problems.⁴³ A review of data-oblivious subspace embeddings for RandNLA, including its relationship with the early work on least absolute deviations regression,¹¹ has been provided.⁵¹ Due to the connection with data-aware and data-oblivious subspace embeddings, approximating matrix multiplication is one of most powerful primitives in RandNLA. Many error formulae for other problems ultimately boil down to matrix inequalities, where the randomness of the algorithm only appears as a (randomized) approximate matrix multiplication.

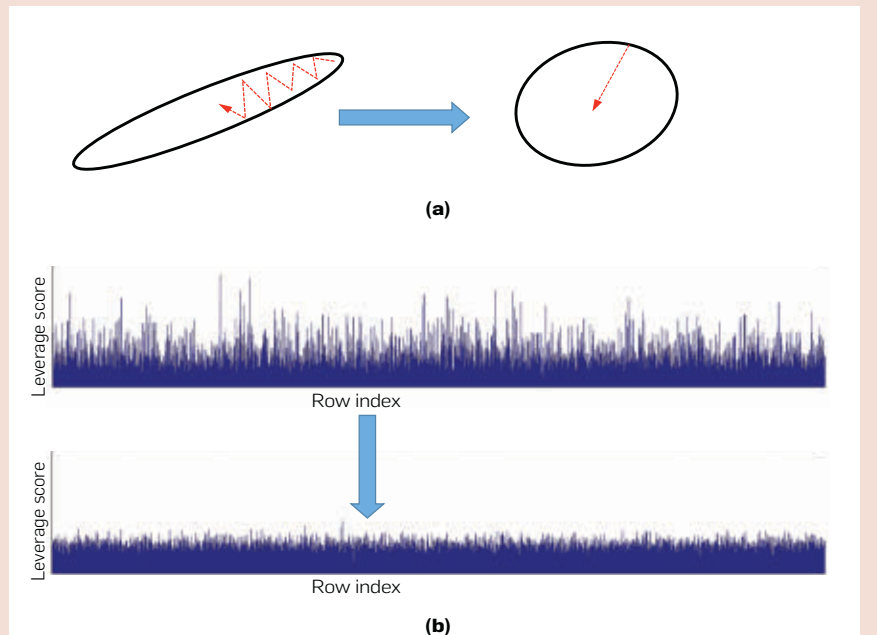
Random projections as preconditioners. Preconditioning refers to

the application of a transformation, called the preconditioner, to a given problem instance such that the transformed instance is more easily solved by a given class of algorithms.^f The main challenge for sampling-based RandNLA algorithms is the construction of the nonuniform sampling probabilities. A natural question arises: is there a way to precondition an input instance such that *uniform random sampling* of rows, columns, or elements yields an insignificant loss in approximation accuracy?

The obvious obstacle to sampling uniformly at random from a matrix is that the relevant information in the matrix could be concentrated on a small number of rows, columns, or elements of the matrix. The solution is to *spread out* or *uniformize* this information, so that it is distributed almost uniformly over all rows, columns, or elements of the matrix. (This is illustrated in Figure 2.) At the same time, the preprocessed

^f For example, if one is interested in iterative algorithms for solving the linear system $Ax = b$, one typically transforms a given problem instance to a related instance in which the so-called condition number is not too large.

Figure 2. In RandNLA, random projections can be used to “precondition” the input data so that uniform sampling algorithms perform well, in a manner analogous to how traditional pre-conditioners transform the input to decrease the usual condition number so that iterative algorithms perform well (see (a)). In RandNLA, the random projection-based preconditioning involves uniformizing information in the eigenvectors, rather than flattening the eigenvalues (see (b)).



matrix should have similar properties (for example, singular values and singular vectors) as the original matrix, and the preprocessing should be computationally efficient (for example, it should be faster than solving the original problem exactly) to perform.

Consider Algorithm 3, our meta-algorithm for preprocessing an input matrix A in order to uniformize information in its rows or columns or elements. Depending on the choice of preprocessing (only from the left, only from the right, or from both sides) the information in A is uniformized in different ways (across its rows, columns, or elements, respectively). For pedagogical simplicity, Algorithm 3 is described such that the output matrix has the same dimensions as the original matrix (in which case Π is approximately a *random rotation*). Clearly, however, if this algorithm is coupled with Algorithm 1 or Algorithm 2, then with trivial to implement uniform sampling, only the rows/columns that are sampled actually need to be generated. In this case the sampled version of Π is known as a *random projection*.

Algorithm 3 A meta-algorithm for preconditioning a matrix for random sampling algorithms

1: **Input:** $m \times n$ matrix A , randomized preprocessing matrices Π_L and/or Π_R .

2: **Output:**

- To uniformize information across the rows of A , return $\Pi_L A$.
 - To uniformize information across the columns of A , return $A \Pi_R$.
 - To uniformize information across the elements of A , return $\Pi_L A \Pi_R$.
-

There is wide latitude in the choice of the random matrix Π . For example, although Π can be chosen to be a random orthogonal matrix, other constructions can have much better algorithmic properties: Π can consist of appropriately-scaled independent identically distributed (i.i.d.) Gaussian random variables, i.i.d. Rademacher random variable (+1 or -1, up to scaling, each with probability 50%), or i.i.d. random variables drawn from any sub-Gaussian distribution. Implementing these variants depends on the time to generate the random bits plus the time to perform the

matrix-matrix multiplication that actually performs the random projection.

More interestingly, Π could be a so-called Fast Johnson Lindenstrauss Transform (FJLT). This is the product of two matrices, a random diagonal matrix with +1 or -1 on each diagonal entry, each with probability 1/2, and the Hadamard-Walsh (or related Fourier-based) matrix.³ Implementing FJLT-based random projections can take advantage of well-studied fast Fourier techniques and can be extremely fast for arbitrary dense input matrices.^{4,41} Recently, there has even been introduced an extremely sparse random projection construction that for arbitrary input matrices can be implemented in “input-sparsity time,” that is, time depending on the number of nonzeros of A , plus lower-order terms, as opposed to the dimensions of A .^{12,34,36}

With appropriate settings of problem parameters (for example, the number of uniform samples that are subsequently drawn, which equals the dimension onto which the data is projected), all of these methods precondition arbitrary input matrices so that uniform sampling in the randomly rotated basis performs as well as non-uniform sampling in the original basis. For example, if $m \gg n$, in which case the leverage scores of A are given by Equation (6), then by keeping only roughly $O(n \log n)$ randomly-rotated dimensions, uniformly at random, one can prove that the leverage scores of the preconditioned system are, up to logarithmic fluctuations, uniform.^g Which construction for Π should be used in any particular application of RandNLA depends on the details of the problem, for example, the aspect ratio of the matrix, whether the RAM model is appropriate for the particular computational infrastructure, how expensive it is to generate random bits, and so on. For example, while slower in the RAM model, Gaussian-based random projections can have stronger conditioning properties than other constructions. Thus, given their ease of use, they are often more appropriate for certain parallel and cloud-computing architectures.^{25,35}

Summary. Of the three basic RandNLA principles described in this section, the

first two have to do with identifying non-uniformity structure in the input data; and the third has to do with preconditioning the input (that is, uniformizing the nonuniformity structure) so uniform random sampling performs well. Depending on the area in which RandNLA algorithms have been developed and/or implemented and/or applied, these principles can manifest themselves in very different ways. Relatedly, in applications where elements are of primary importance (for example, recommender systems²⁶), element-wise methods might be most appropriate, while in applications where subspaces are of primary importance (for example, scientific computing²⁵), column/row-based methods might be most appropriate.

Extensions and Applications of Basic RandNLA Principles

We now turn to several examples of problems in various domains where the basic RandNLA principles have been used in the design and analysis, implementation, and application of novel algorithms.

Low-precision approximations and high-precision numerical implementations: least-squares and low-rank approximation. One of the most fundamental problems in linear algebra is the least-squares (LS) regression problem: given an $m \times n$ matrix A and an m -dimensional vector b , solve

$$x_{opt} = \arg \min_{x \in \mathbb{R}^n} \|Ax - b\|_2, \quad (8)$$

where $\|\cdot\|_2$ denotes the ℓ_2 norm of a vector. That is, compute the n -dimensional vector x that minimizes the Euclidean norm of the residual $Ax - b$.^h If $m \gg n$, then we have the overdetermined (or overconstrained) LS problem, and its solution can be obtained in $O(mn^2)$ time in the RAM model with one of several methods, for example, solving the normal equations, QR decompositions, or the SVD. Two major successes of RandNLA concern faster (in terms of low-precision asymptotic worst-case theory, or in terms of high-precision wall-clock time) algorithms for this ubiquitous problem.

^h Observe this formulation includes as a special case the problem of solving systems of linear equations (if $m = n$ and A has full rank, then the resulting system of linear equations has a unique solution).

^g This is equivalent to the statement that the coherence of the preconditioned system is small.

One major success of RandNLA was the following random sampling algorithm for the LS problem: quickly compute $1 \pm \epsilon$ approximations to the leverage scores;¹⁴ form a subproblem by sampling with Algorithm 2 roughly $\Theta(n \log(m)/\epsilon)$ rows from A and the corresponding elements from b using those approximations as importance sampling probabilities; and return the LS solution of the subproblem.^{14,15} Alternatively, one can run the following random projection algorithm: precondition the input with a Hadamard-based random projection; form a subproblem by sampling with Algorithm 2 roughly $\Theta(n \log(m)/\epsilon)$ rows from A and the corresponding elements from b uniformly at random; and return the LS solution of the subproblem.^{17,43}

Both of these algorithms return $1 \pm \epsilon$ relative-error approximate solutions for arbitrary or worst-case input; and both run in roughly $\Theta(mn \log(n)/\epsilon) = o(mn^2)$ time, that is, qualitatively faster than traditional algorithms for the overdetermined LS problem. (Although this random projection algorithm is not faster in terms of

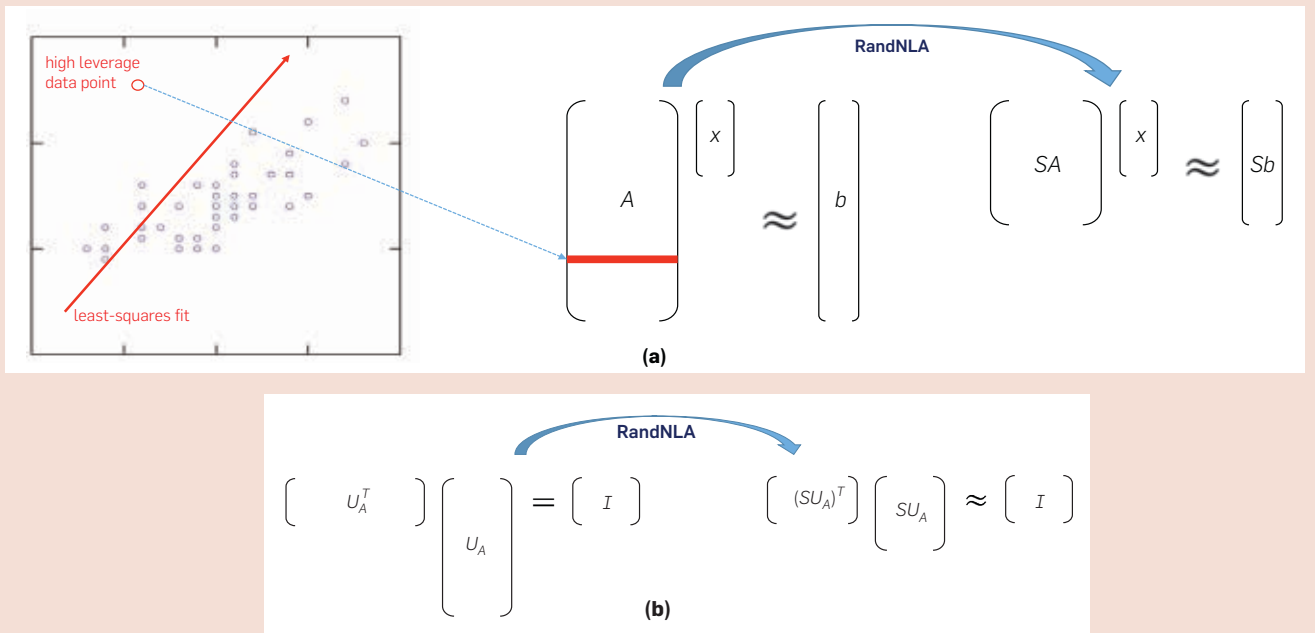
asymptotic FLOPS than the corresponding random sampling algorithm, preconditioning with random projections is a powerful primitive more generally for RandNLA algorithms.) Moreover, both of these algorithms have been improved to run in time that is proportional to the number of nonzeros in the matrix, plus lower-order terms that depend on the lower dimension of the input.¹²

Another major success of RandNLA was the demonstration that the sketches constructed by RandNLA could be used to construct preconditioners for high-quality traditional NLA iterative software libraries.⁴ To see the need for this, observe that because of its dependence on ϵ , the previous RandNLA algorithmic strategy (construct a sketch and solve a LS problem on that sketch) can yield low-precision solutions, for example, $\epsilon = 0.1$, but cannot practically yield high-precision solutions, for example, $\epsilon = 10^{-16}$. Blendenpik⁴ and LSRN³⁵ are LS solvers that are appropriate for RAM and parallel environments, respectively, that adopt the following RandNLA algorithmic strategy: construct a sketch, using

an appropriate random projection; use that sketch to construct a preconditioner for a traditional iterative NLA algorithm; and use that to solve the preconditioned version of the original full problem. This improves the ϵ dependence from $\text{poly}(1/\epsilon)$ to $\log(1/\epsilon)$. Carefully-engineered implementations of this approach are competitive with or beat high-quality numerical implementations of LS solvers such as those implemented in LAPACK.⁴

The difference between these two algorithmic strategies (see Figure 3 for an illustration) highlights important differences between TCS and NLA approaches to RandNLA, as well as between computer science and scientific computing more generally: subtle but important differences in problem parameterization, between what counts as a “good” solution, and between error norms of interest. Moreover, similar approaches have been used to extend TCS-style RandNLA algorithms for providing $1 \pm \epsilon$ relative-error low-rank matrix approximation^{16,43} to NLA-style RandNLA algorithms for high-quality numerical low-rank matrix approximation.^{24,25,41}

Figure 3. (a) RandNLA algorithms for least-squares problems first compute sketches, SA and Sb , of the input data, A and b . Then, either they solve a least-squares problem on the sketch to obtain a low-precision approximation, or they use the sketch to construct a traditional preconditioner for an iterative algorithm on the original input data to get high-precision approximations. Subspace-preserving embedding: if S is a random sampling matrix, then the high leverage point will be sampled and included in SA ; and if S is a random-projection-type matrix, then the information in the high leverage point will be homogenized or uniformized in SA . (b) The “heart” of RandNLA proofs is subspace-preserving embedding for orthogonal matrices: if U_A is an orthogonal matrix (say the matrix of the left singular vectors of A), then SU_A is approximately orthogonal.



For example, a fundamental structural condition for a sketching matrix to satisfy to obtain good low-rank matrix approximation is the following. Let $V_k \in \mathbb{R}^{n \times k}$ (resp., $V_{k,\perp} \in \mathbb{R}^{n \times (n-k)}$) be any matrix spanning the top- k (resp., bottom- $(n-k)$) right singular subspace of $A \in \mathbb{R}^{m \times n}$, and let Σ_k (resp., $\Sigma_{k,\perp}$) be the diagonal matrix containing the top- k (resp., all but the top- k) singular values. In addition, let $Z \in \mathbb{R}^{n \times r}$ ($r \geq k$) be any matrix (for example, a random sampling matrix S , a random projection matrix Π , or a matrix Z constructed deterministically) such that $V_k^T Z$ has full rank. Then,

$$\|A - P_{AZ}A\| \leq \|\Sigma_k\| + \left\| \Sigma_{k,\perp} \left(V_{k,\perp}^T Z \right) \left(V_k^T Z \right)^+ \right\|, \quad (9)$$

where $\|\cdot\|$ is any unitarily invariant matrix norm.

How this structural condition is used depends on the particular low-rank problem of interest, but it is widely used (either explicitly or implicitly) by low-rank RandNLA algorithms. For example, Equation (9) was introduced in the context of the Column Subset Selection Problem⁷ and was reproven and used to reparameterize low-rank random projection algorithms in ways that could be more easily implemented.²⁵ It has also been used in ways ranging from developing improved bounds for kernel methods in machine learning²¹ to coupling with a version of the power method to obtain improved numerical implementations⁴¹ to improving subspace iteration methods.²⁴

The structural condition in Equation (9) immediately suggests a proof strategy for bounding the error of RandNLA algorithms for low-rank matrix approximation: identify a sketching matrix Z such that $V_k^T Z$ has full rank; and, at the same time, bound the relevant norms of $(V_k^T Z)^+$ and $V_{k,\perp}^T Z$. Importantly, in many of the motivating scientific computing applications, the matrices of interest are linear operators that are only implicitly represented but that are structured such that they can be applied to an arbitrary vector quickly. In these cases, FJLT-based or input-sparsity-based projections applied to arbitrary matrices can be replaced with Gaussian-based projections applied to these structured operators with similar computational costs and quality guarantees.

Matrix completion. Consider the

following problem, which is an idealization of the important recommender systems problem.²⁶ Given an arbitrary $m \times n$ matrix A , reconstruct A by sampling a set of $O((m+n)\text{poly}(1/\epsilon^\alpha))$, as opposed to all mn , entries of the matrix such that the resulting approximation $\tilde{A}$ satisfies, either deterministically or up to some failure probability,

$$\|A - \tilde{A}\|_F \leq (1 + \epsilon) \|A - A_k\|_F. \quad (10)$$

Here, α should be small (for example, 2); and the sample size could be increased by (less important) logarithmic factors of m , n , and ϵ . In addition, one would like to construct the sample and compute $\tilde{A}$ after making a small number of passes over A or without even touching all of the entries of A .

A first line of research (already mentioned) on this problem from TCS focuses on element-wise sampling:² sample entries from a matrix with probabilities that (roughly) depend on their magnitude squared. This can be done in one pass over the matrix, but the resulting additive-error bound is much larger than the requirements of Equation (10), as it scales with the Frobenius norm of A instead of the Frobenius norm of $A - A_k$.

A second line of research from signal processing and applied mathematics has referred to this as the *matrix completion problem*.⁸ In this case, one is interested in computing $\tilde{A}$ without even observing all of the entries of A . Clearly, this is not possible without assumptions on A .ⁱ Typical assumptions are on the eigenvalues and eigenvectors of A : for example, the input matrix A has rank exactly k , with $k \ll \min\{m, n\}$, and also that A satisfies some sort of *eigenvector delocalization* or *incoherence conditions*.⁸ The simplest form of the latter is the leverage scores of Equation (6) are approximately uniform. Under these assumptions, one can prove that given a uniform sample of $O((m+n)k \ln(m+n))$ entries of A , the solution to the following nuclear norm minimization problem recovers A exactly, with high probability:

i This highlights an important difference in problem parameterization: TCS-style approaches assume worst-case input and must identify nonuniformity structure, while applied mathematics approaches typically assume well-posed problems where the worst nonuniformity structure is not present.

$$\min_{\tilde{A} \in \mathbb{R}^{m \times n}} \|\tilde{A}\|_* \quad (11)$$

s.t. $\tilde{A}_{ij} = A_{ij}$, for all sampled entries A_{ij} ,

where $\|\cdot\|_*$ denotes the nuclear (or trace) norm of a matrix (basically, the sum of the singular values of the matrix). That is, if A is exactly low-rank (that is, $A = A_k$ and thus $A - A_k$ is zero) and satisfies an incoherence assumption, then Equation (10) is satisfied, since $A = A_k = \tilde{A}$. Recently, the incoherence assumption has been relaxed, under the assumption that one is given oracle access to A according to a non-uniform sampling distribution that essentially corresponds to *element-wise leverage scores*.¹⁰ However, removing the assumption that A has exact low-rank k , with $k \ll \min\{m, n\}$, is still an open problem.^j

Informally, keeping only a few rows/columns of a matrix seems more powerful than keeping a comparable number of elements of a matrix. For example, consider an $m \times n$ matrix A whose rank is exactly equal to k , with $k \ll \min\{m, n\}$: selecting any set of k linearly independent rows allows every row of A to be expressed as a linear combination of the selected rows. The analogous procedure for element-wise sampling seems harder. This is reflected in that state-of-the-art element-wise sampling algorithms use convex optimization and other heavier-duty algorithmic machinery.

Solving systems of Laplacian-based linear equations. Consider the special case of the LS regression problem of Equation (8) when $m = n$, that is, the well-known problem of solving the system of linear equations $Ax = b$. For worst-case dense input matrices A this problem can be solved in exactly $O(n^3)$ time, for example, using the partial LU decomposition and other methods. However, especially when A is positive semidefinite (PSD), iterative techniques such as the conjugate gradients method are typically preferable, mainly because of their linear dependency on the number of non-zero entries in the matrix A (times a factor depending on the condition number of A).

An important special case is when the PSD matrix A is the Laplacian matrix

j It should be noted that there exists prior work on matrix completion for low-rank matrices with the addition of well-behaved noise; however, removing the low-rank assumption and achieving error that is relative to some norm of the residual $A - A_k$ is still open.

of an underlying undirected graph $G = (V, E)$, with $n = |V|$ vertices and $|E|$ weighted, undirected edges.⁵ Variants of this special case are common in unsupervised and semi-supervised machine learning.⁶ Recall the Laplacian matrix of an undirected graph G is an $n \times n$ matrix that is equal to the $n \times n$ diagonal matrix D of node weights minus the $n \times n$ adjacency matrix of the graph. In this special case, there exist randomized, relative-error algorithms for the problem of Equation (8).⁵ The running time of these algorithms is

$$O(\text{nnz}(A)\text{polylog}(n)),$$

where $\text{nnz}(A)$ represents the number of non-zero elements of the matrix A , that is, the number of edges in the graph G . The first step of these algorithms corresponds to randomized graph sparsification and keeps a small number of edges from G , thus creating a much sparser Laplacian matrix $\tilde{L}$. This sparse matrix $\tilde{L}$ is subsequently used (in a recursive manner) as an efficient preconditioner to approximate the solution of the problem of Equation (8).

While the original algorithms in this line of work were major theoretical breakthroughs, they were not immediately applicable to numerical implementations and data applications. In an effort to bridge the theory-practice gap, subsequent work proposed a much simpler algorithm for the graph sparsification step.⁴⁵ This subsequent work showed that randomly sampling edges from the graph G (equivalently, rows from the edge-incidence matrix) with probabilities proportional to the *effective resistances* of the edges provides a sparse Laplacian matrix $\tilde{L}$ satisfying the desired properties. (On the negative side, in order to approximate the effective resistances of the edges of G , a call to the original solver was necessary, clearly hindering the applicability of the simpler sparsification algorithm.⁴⁵) The effective resistances are equivalent to the statistical leverage scores of the weighted edge-incidence matrix of G . Subsequent work has exploited graph theoretic ideas to provide efficient algorithms to approximate them in time proportional to the number of edges in the graph (up to polylogarithmic factors).²⁷ Recent improvements have essentially



RandNLA has proven to be a model for truly interdisciplinary research in this era of large-scale data.



removed these polylogarithmic factors, leading to useful implementations of Laplacian-based solvers.²⁷ Extending such techniques to handle general PSD input matrices A that are not Laplacian is an open problem.

Statistics and machine learning. RandNLA has been used in statistics and machine learning in several ways, the most common of which is in the so-called kernel-based machine learning.²¹ This involves using a PSD matrix to encode nonlinear relationships between data points; and one obtains different results depending on whether one is interested in approximating a given kernel matrix,²¹ constructing new kernel matrices of particular forms,³⁹ or obtaining a low-rank basis with which to perform downstream classification, clustering, and other related tasks.²⁹ Alternatively, the analysis used to provide relative-error low-rank matrix approximation for worst-case input can also be used to provide bounds for kernel-based divide-and-conquer algorithms.³¹ More generally, CX/CUR decompositions provide scalable and interpretable solutions to downstream data analysis problems in genetics, astronomy, and related areas.^{33,38,53,54} Recent work has focused on statistical aspects of the “algorithmic leveraging” approach that is central to RandNLA algorithms.³⁰

Looking Forward

RandNLA has proven to be a model for truly interdisciplinary research in this era of large-scale data. For example, while TCS, NLA, scientific computing, mathematics, machine learning, statistics, and downstream scientific domains are all interested in these results, each of these areas is interested for very different reasons. Relatedly, while technical results underlying the development of RandNLA have been nontrivial, some of the largest obstacles to progress in RandNLA have been cultural: TCS being cavalier about polynomial factors, ϵ factors, and working in overly idealized computational models; NLA being extremely slow to embrace randomization as an algorithmic resource; scientific computing researchers formulating and implementing algorithms that make strong domain-specific assumptions; and machine learning and statistics researchers being more interested in results on hypoth-

sized unseen data rather than the data being input to the algorithm.

In spite of this, RandNLA has already led to improved algorithms for several fundamental matrix problems, but it is important to emphasize that “improved” means different things to different people. For example, TCS is interested in these methods due to the deep connections with Laplacian-based linear equation solvers^{5,27} and since fast random sampling and random projection algorithms^{12,14,17,43} represent an improvement in the asymptotic running time of the 200-year-old Gaussian elimination algorithms for least-squares problems on worst-case input. NLA is interested in these methods since they can be used to engineer variants of traditional NLA algorithms that are more robust and/or faster in wall clock time than high-quality software that has been developed over recent decades. (For example, Blendenpik “beats LAPACK’s direct dense least-squares solver by a large margin on essentially any dense tall matrix;”⁴ the randomized approach for low-rank matrix approximation in scientific computing “beats its classical competitors in terms of accuracy, speed, and robustness;”²⁵ and least-squares and least absolute deviations regression problems “can be solved to low, medium, or high precision in existing distributed systems on up to terabyte-sized data.”⁵²) Mathematicians are interested in these methods since they have led to new and fruitful fundamental mathematical questions.^{23,40,42,47} Statisticians and machine learners are interested in these methods due to their connections with kernel-based learning and since the randomness inside the algorithm often implicitly implements a form of regularization on realistic noisy input data.^{21,29,30} Finally, data analysts are interested in these methods since they provide scalable and interpretable solutions to downstream scientific data analysis problems.^{33,38,54} Given the central role that matrix problems have historically played in large-scale data analysis, we expect RandNLA methods will continue to make important contributions not only to each of those research areas but also to bridging the gaps between them. 

References

- Achlioptas, D., Karnin, Z., Liberty, E. Near-optimal entropy sampling for data matrices. In *Annual Advances in Neural Information Processing Systems 26: Proceedings of the 2013 Conference*, 2013.
- Achlioptas, D., McSherry, F. Fast computation of low-rank matrix approximations. *J. ACM* 54, 2 (2007), Article 9.
- Ailon, N., Chazelle, B. Faster dimension reduction. *Commun. ACM* 53, 2 (2010), 97–104.
- Avron, H., Maymounkov, P., Toledo, S. Blendenpik: Supercharging LAPACK’s least-squares solver. *SIAM J. Sci. Comput.* 32 (2010), 1217–1236.
- Batson, J., Spielman, D.A., Srivastava, N., Teng, S.-H. Spectral sparsification of graphs: Theory and algorithms. *Commun. ACM* 56, 8 (2013), 87–94.
- Belkin, M., Niyogi, P. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Comput.* 15, 6 (2003), 1373–1396.
- Boutsidis, C., Mahoney, M.W., Drineas, P. An improved approximation algorithm for the column subset selection problem. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms* (2009), 968–977.
- Candes, E.J., Recht, B. Exact matrix completion via convex optimization. *Commun. ACM* 55, 6 (2012), 111–119.
- Chatterjee, S., Hadi, A.S. Influential observations, high leverage points, and outliers in linear regression. *Stat. Sci.* 1, 3 (1986), 379–393.
- Chen, Y., Bhojanapalli, S., Sanghavi, S., Ward, R. Coherent matrix completion. In *Proceedings of the 31st International Conference on Machine Learning* (2014), 674–682.
- Clarkson, K. Subgradient and sampling algorithms for ℓ_1 regression. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms* (2005), 257–266.
- Clarkson, K.L., Woodruff, D.P. Low rank approximation and regression in input sparsity time. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing* (2013), 81–90.
- Drineas, P., Kannan, R., Mahoney, M.W. Fast Monte Carlo algorithms for matrices I: approximating matrix multiplication. *SIAM J. Comput.* 36 (2006), 132–157.
- Drineas, P., Magdon-Ismail, M., Mahoney, M.W., Woodruff, D.P. Fast approximation of matrix coherence and statistical leverage. *J. Mach. Learn. Res.* 13 (2012), 3475–3506.
- Drineas, P., Mahoney, M.W., Muthukrishnan, S. Sampling algorithms for ℓ_1 regression and applications. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms* (2006), 1127–1136.
- Drineas, P., Mahoney, M.W., Muthukrishnan, S. Relative-error CUR matrix decompositions. *SIAM J. Matrix Anal. Appl.* 30 (2008), 844–881.
- Drineas, P., Mahoney, M.W., Muthukrishnan, S., Sarlós, T. Faster least squares approximation. *Numer. Math.* 117, 2 (2010), 219–249.
- Drineas, P., Zouzias, A. A note on element-wise matrix sparsification via a matrix-valued Bernstein inequality. *Inform. Process. Lett.* 111 (2011), 385–389.
- Frieze, A., Kannan, R., Vempala, S. Fast Monte-Carlo algorithms for finding low-rank approximations. *J. ACM* 51, 6 (2004), 1025–1041.
- Füredi, Z., Komlós, J. The eigenvalues of random symmetric matrices. *Combinatorica* 1, 3 (1981), 233–241.
- Gittens, A., Mahoney, M.W. Revisiting the Nyström method for improved large-scale machine learning. *J. Mach. Learn. Res.* In press.
- Golub, G.H., Van Loan, C.F. *Matrix Computations*. Johns Hopkins University Press, Baltimore, 1996.
- Gross, D. Recovering low-rank matrices from few coefficients in any basis. *IEEE Trans. Inform. Theory* 57, 8 (2011), 1548–1566.
- Gu, M. Subspace iteration randomization and singular value problems. Technical report, 2014. Preprint: arXiv:1408.2208.
- Halko, N., Martinsson, P.-G., Tropp, J.A. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev.* 53, 2 (2011), 217–288.
- Koren, Y., Bell, R., Volinsky, C. Matrix factorization techniques for recommender systems. *IEEE Comp.* 42, 8 (2009), 30–37.
- Koutis, I., Miller, G.L., Peng, R. A fast solver for a class of linear systems. *Commun. ACM* 55, 10 (2012), 99–107.
- Kundu, A., Drineas, P. A note on randomized element-wise matrix sparsification. Technical report, 2014. Preprint: arXiv:1404.0320.
- Le, Q.V., Sarlós, T., Smola, A.J. Fastfood—approximating kernel expansions in loglinear time. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.
- Ma, P., Mahoney, M.W., Yu, B. A statistical perspective on algorithmic leveraging. *J. Mach. Learn. Res.* 16 (2015), 861–911.
- Mackey, L., Talwalkar, A., Jordan, M.I. Distributed matrix completion and robust factorization. *J. Mach. Learn. Res.* 16 (2015), 913–960.
- Mahoney, M.W. *Randomized Algorithms for Matrices and Data*. Foundations and Trends in Machine Learning. NOW Publishers, Boston, 2011.
- Mahoney, M.W., Drineas, P. CUR matrix decompositions for improved data analysis. *Proc. Natl. Acad. Sci. USA* 106 (2009), 697–702.
- Meng, X., Mahoney, M.W. Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing* (2013), 91–100.
- Meng, X., Saunders, M.A., Mahoney, M.W. LSRN: A parallel iterative solver for strongly over- or under-determined systems. *SIAM J. Sci. Comput.* 36, 2 (2014), C95–C118.
- Nelson, J., Huy, N.L. OSNAP: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science* (2013), 117–126.
- Oliveira, R.I. Sums of random Hermitian matrices and an inequality by Rudelson. *Electron. Commun. Probab.* 15 (2010) 203–212.
- Paschou, P., Ziv, E., Burchard, E.G., Choudhry, S., Rodriguez-Cintrón, W., Mahoney, M.W., Drineas, P. PCA-correlated SNPs for structure identification in worldwide human populations. *PLoS Genet.* 3 (2007), 1672–1686.
- Rahimi, A., Recht, B. Random features for large-scale kernel machines. In *Annual Advances in Neural Information Processing Systems 20: Proceedings of the 2007 Conference*, 2008.
- Recht, B. A simpler approach to matrix completion. *J. Mach. Learn. Res.* 12 (2011), 3413–3430.
- Rokhlin, V., Szlam, A., Tytgert, M. A randomized algorithm for principal component analysis. *SIAM J. Matrix Anal. Appl.* 31, 3 (2009), 1100–1124.
- Rudelson, M., Vershynin, R. Sampling from large matrices: an approach through geometric functional analysis. *J. ACM* 54, 4 (2007), Article 21.
- Sarlós, T. Improved approximation algorithms for large matrices via random projections. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science* (2006), 143–152.
- Smale, S. Some remarks on the foundations of numerical analysis. *SIAM Rev.* 32, 2 (1990), 211–220.
- Spielman, D.A., Srivastava, N. Graph sparsification by effective resistances. *SIAM J. Comput.* 40, 6 (2011), 1913–1926.
- Stigler, S.M. *The History of Statistics: The Measurement of Uncertainty before 1900*. Harvard University Press, Cambridge, 1986.
- Tropp, J.A. User-friendly tail bounds for sums of random matrices. *Found. Comput. Math.* 12, 4 (2012), 389–434.
- Turing, A.M. Rounding-off errors in matrix processes. *Quart. J. Mech. Appl. Math.* 1 (1948), 287–308.
- von Neumann, J., Goldstine, H.H. Numerical inverting of matrices of high order. *Bull. Am. Math. Soc.* 53 (1947), 1021–1099.
- Wigner, E.P. Random matrices in physics. *SIAM Rev.* 9, 1 (1967), 1–23.
- Woodruff, D.P. *Sketching as a Tool for Numerical Linear Algebra*. Foundations and Trends in Theoretical Computer Science. NOW Publishers, Boston, 2014.
- Yang, J., Meng, X., Mahoney, M.W. Implementing randomized matrix algorithms in parallel and distributed environments. *Proc. IEEE* 104, 1 (2016), 58–92.
- Yang, J., Rüböl, O., Prabhat, Mahoney, M.W., Bowen, B.P. Identifying important ions and positions in mass spectrometry imaging data using CUR matrix decompositions. *Anal. Chem.* 87, 9 (2015), 4658–4666.
- Yip, C.-W., Mahoney, M.W., Szalay, A.S., Csabai, I., Budavari, T., Wyse, R.F.G., Dobos, L. Objective identification of informative wavelength regions in galaxy spectra. *Astron. J.* 147, 110 (2014), 15.

Petros Drineas (drineas@gmail.com) is an associate professor in the Department of Computer Science at Rensselaer Polytechnic Institute, Troy, NY.

Michael W. Mahoney (mmahoney@stat.berkeley.edu) is an associate professor in ICSI and in the Department of Statistics at the University of California at Berkeley.

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

research highlights

P. 92

**Technical
Perspective**
**Veritesting Tackles
Path-Explosion
Problem**

By Koushik Sen

P. 93

Enhancing Symbolic Execution with Veritesting

By Thanassis Avgerinos, Alexandre Rebert,
Sang Kil Cha, and David Brumley

P. 101

**Technical
Perspective**
**Computing with
the Crowd**

By Siddharth Suri

P. 102

AutoMan: A Platform for Integrating Human-Based and Digital Computation

By Daniel W. Barowy, Charlie Curtsinger,
Emery D. Berger, and Andrew McGregor

Technical Perspective

Veritestng Tackles Path-Explosion Problem

By Koushik Sen

IMAGINE YOU ARE working on a large piece of software for a safety-critical system, such as the braking system of a car. How would you make sure the car will not accelerate under any circumstance when the driver applies the brake? How would you know that someone other than the driver would not be able to stop a moving car by exploiting a remote security vulnerability in the software system? How would you confirm the braking system will not fail suddenly due to a fatal crash in the software system?

Testing is the only predominant technique used by the software industry to answer such questions and to make software systems reliable. Studies show that testing accounts for more than half of the total software development cost in industry. Although testing is a widely used and a well-established technique for building reliable software, existing techniques for testing are mostly ad hoc and ineffective—serious bugs are often exposed post-deployment. Wouldn't it be nice if one could build a software system that could exhaustively test any software and report all critical bugs in the software to its developer?

In recent years, symbolic execution has emerged as one such automated technique to generate high-coverage test suites. Such test suites could find deep errors and security vulnerabilities in complex software applications. Symbolic execution analyzes the source code or the object code of a program to determine what inputs would execute the different paths of the program. The key idea behind symbolic execution was introduced almost 40 years ago. However, it has only recently been made practical, as a result of significant advances in program analysis and constraint-solving techniques, and due to the invention of *dynamic symbolic execution (DSE)* or *concolic testing*, which combines concrete and symbolic execution.

Since its introduction in 2005, DSE and concolic testing have inspired the development of several scalable symbolic execution tools such as DART, CUTE, jCUTE, KLEE, JPF, SAGE, PEX, CREST, BitBlaze, S2E, Jalangi, CATG, Triton, CONBOL, and SymDroid. Such tools have been used to find crashing inputs, to generate high-coverage test-suites, and to expose security vulnerabilities. For example, Microsoft's SAGE has discovered one-third of all bugs revealed during the development of Windows 7.

Although modern symbolic execution tools have been successful in finding high-impact bugs and security vulnerabilities, it has been observed that symbolic execution techniques do not scale well to large realistic programs because the number of feasible execution paths of a program often increases exponentially with the length of an execution path. Therefore, most modern symbolic execution tools achieve poor coverage when they are applied to large programs. Most of the research in symbolic execution nowadays is, therefore, focusing on mitigating the *path-explosion* problem.

To mitigate the path-explosion problem, a number of techniques have been proposed to merge symbolic execution paths at various program points. Symbolic path merging, also known as *static symbolic execution (SSE)*, enables carrying out symbolic execution of multiple paths simultaneously. However, this form of path merging often leads to large and complex formula that are difficult to solve. Moreover, path merging fails to work for real-world programs that perform system calls. Despite these recent proposals for mitigating the path explosion problem, the proposed techniques are not effective enough to handle large systems code.

The following work by Avgerinos et al. is a landmark in further addressing the path-explosion problem for real-

world software systems. The authors have proposed an effective technique called veritestng that addresses the scalability limitations of path merging in symbolic execution. They have implemented veritestng in Merge-Point, a tool for automatically testing all program binaries in a Linux distribution. A key attraction of Merge-Point is that the tool can be applied to any binary without any source information or re-compilation or preprocessing or user-setup. A broader impact of this work is that users can now apply symbolic execution to larger software systems and achieve better code coverage while finding deep functional and security bugs.

Veritestng works by alternating between dynamic symbolic execution and path merging or *static symbolic execution*. DSE helps to handle program fragments that cannot be handled by SSE, such as program fragments making system calls and indirect jumps. SSE, on the other hand, helps to avoid repeated exploration of exponential number of paths in small program fragments by summarizing their behavior as a formula. What I find truly remarkable is this clever combination of DSE and SSE has enabled veritestng to scale to thousands of binaries in a Linux distribution. The tool has found more than 10,000 bugs in the distribution and Debian maintainers have already applied patches to 229 such bugs. These results and impact on real-world software have demonstrated that symbolic execution has come out of its infancy and has become a viable alternative for testing real-world software systems without user-intervention. 

Koushik Sen (ksen@cs.berkeley.edu) is an associate professor in the Department of Electrical Engineering and Computer Sciences at the University of California, Berkeley.

Copyright held by author.

Enhancing Symbolic Execution with Veritesting

By Thanassis Avgerinos, Alexandre Rebert, Sang Kil Cha, and David Brumley

1. INTRODUCTION

Symbolic execution is a popular automatic approach for testing software and finding bugs. Over the past decade, numerous symbolic execution tools have appeared—both in academia and industry—demonstrating the effectiveness of the technique in finding crashing inputs, generating test cases with high coverage, and exposing software vulnerabilities.⁵ Microsoft's symbolic executor SAGE is responsible for finding *one-third* of all bugs discovered during the development of Windows 7.¹²

Symbolic execution is attractive because of two salient features. First, it generates *real* test cases; every bug report is accompanied by a concrete input that reproduces the problem (thus eliminating false reports). Second, symbolic execution systematically checks each program path exactly once—no work will be repeated as in other typical testing techniques (e.g., random fuzzing).

Symbolic execution works by automatically translating program fragments to logical formulas. The logical formulas are satisfied by inputs that have a desired property, for example, they execute a specific path or violate safety. Thus, with symbolic execution, finding crashing test cases effectively reduces to finding satisfying variable assignments in logical formulas, a process typically automated by Satisfiability Modulo Theories (SMT) solvers.⁹

At a high level, there are two main approaches for generating formulas: dynamic symbolic execution (DSE) and static symbolic execution (SSE). DSE executes the analyzed program fragment and generates formulas on a per-path basis. SSE translates program fragments into formulas, where each formula represents the desired property over *any* path within the selected fragment. The path-based nature of DSE introduces significant overhead when generating formulas, but the formulas themselves are easy to solve. The statement-based nature of SSE has less overhead and produces more succinct formulas that cover more paths, but the formulas are harder to solve. Is there a way to get the best of both worlds?

In this article, we present a new technique for generating formulas called *veritesting* that *alternates* between SSE and DSE. The alternation mitigates the difficulty of solving formulas, while alleviating the high overhead associated with a path-based DSE approach. In addition, DSE systems replicate the path-based nature of concrete execution, allowing them to handle cases such as system calls and indirect jumps where static approaches would need summaries or additional analysis. Alternating allows veritesting to switch to DSE-based methods when such cases are encountered.

We implemented veritesting in MergePoint, a system for automatically checking all programs in a Linux distribution.

MergePoint operates on 32-bit Linux binaries and does not require any source information (e.g., debugging symbols). We have systematically used MergePoint to test and evaluate veritesting on 33,248 binaries from Debian Linux. The binaries were collected by downloading and mining for executable programs all available packages from the Debian main repository. We did not pick particular binaries or a dataset that would highlight specific aspects of our system; instead we focus on our system as experienced in the general case. The large dataset allows us to explore questions with high fidelity and with a smaller chance of per-program sample bias. The binaries are exactly what runs on millions of systems throughout the world.

We demonstrate that MergePoint with veritesting beats previous techniques in the three main metrics: bugs found, node coverage, and path coverage. In particular, MergePoint has found 11,687 distinct bugs in 4379 different programs. Overall, MergePoint has generated over 15 billion SMT queries and created over 200 million test cases. Out of the 1043 bugs we have reported so far to the developers, 229 have been fixed.

Our main contributions are as follows. First, we propose a new technique for symbolic execution called veritesting. Second, we provide and study in depth the first system for testing every binary in an OS distribution using symbolic execution. Our experiments reduce the chance of per-program or per-dataset bias. We evaluate MergePoint with and without veritesting and show that veritesting outperforms previous work on all three major metrics. Finally, we improve open source software by finding over 10,000 bugs and generating millions of test cases. Debian maintainers have already incorporated 229 patches due to our bug reports. We have made our data available on our website.²⁰ For more experiments and details we refer the reader to the original paper.²

2. SYMBOLIC EXECUTION BACKGROUND

Symbolic execution¹⁴ is similar to normal program execution with one main twist: instead of using concrete input values, symbolic execution uses variables (*symbols*). During execution, all program values are expressed in terms of input variables. To keep track of the currently executing path, symbolic execution stores all conditions required to follow the same path (e.g., assertions, conditions on branch statements, etc.) in a logical formula called the path predicate.

The original version of this paper was published in the *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India, May 31–June 7, 2014). ACM, New York, NY, 1083–1094.

Algorithm 1: Dynamic Symbolic Execution Algorithm

Input: Initial program counter (entry point): pc_0
 Instruction fetch & decode: `instrFetchDecode`
Data: State worklist: *Worklist*, path predicate: Π ,
 variable state: Δ

```

1 Function ExecuteInstruction(instr, pc,  $\Pi$ ,  $\Delta$ )
2   switch instr do
3     case var := exp // assignment
4        $\Delta[\text{var}] \leftarrow \text{exp}$ 
5       return [(succ(pc),  $\Pi$ ,  $\Delta$ )]
6     case assert (exp) // assertion
7       return [(succ(pc),  $\Pi \wedge \text{exp}$ ,  $\Delta$ )]
8     case if (exp) goto pc' // conditional jump
9       // Regular DSE forks 2 states
10      return [(pc',  $\Pi \wedge \text{exp}$ ,  $\Delta$ ), (succ(pc),  $\Pi \wedge \neg \text{exp}$ ,
11       $\Delta$ )]
12      // Veritestng integration
13      return Veritest(pc,  $\Pi$ ,  $\Delta$ )
14     case halt: return [] // terminate
15  Worklist = [(pc0, true, {})] // initial worklist
16  while Worklist ≠ [] do
17    pc,  $\Pi$ ,  $\Delta$  = removeOne(Worklist)
18    instr = instrFetchDecode(pc)
19    NewStates = ExecuteInstruction(instr, pc,  $\Pi$ ,  $\Delta$ )
20    Worklist = add(Worklist, NewStates)

```

Inputs that make the path predicate true are guaranteed to follow the exact same execution path. If there is no input satisfying the path predicate, the current execution path is infeasible.

In the following sections, we give a brief overview of the two main symbolic execution approaches: dynamic and SSE. We refer the reader to symbolic execution surveys for more details and examples.^{5, 21}

2.1. Dynamic symbolic execution (DSE)

Algorithm 1 presents the core steps in DSE. The algorithm operates on a representative low-level language with assignments, assertions and conditional jumps (simplified from the original Avgerinos et al.²).

Similar to an interpreter, a symbolic executor consists of a main instruction fetch-decode-execute loop (Lines 14–17). On each iteration, the `removeOne` function selects the next state to execute from *Worklist*, decodes the instruction, executes it and inserts the new execution states in *Worklist*. Each execution state is a triple (*pc*, Π , Δ) where *pc* is the current program counter, Π is the path predicate (the condition under which the current path will be executed), and Δ is a dictionary that maps each variable to its current value.

Unlike concrete interpreters, symbolic executors need to maintain a list of execution states (not only one). The reason is conditional branches. Line 10 (highlighted in red) demonstrates why: the executed branch condition could be true or false—depending on the program input—and the symbolic executor needs to execute *both* paths in order to check correctness. The process of

generating two new execution states out of a single state (one for the true branch and one for the false), is typically called “forking.” Due to forking, every branch encountered during execution, doubles the number of states that need to be analyzed, a problem known in DSE as *path (or state) explosion*.⁵

Example. We now give a short example demonstrating DSE in action. Consider the following program:

```

1   if (input_char == 'B') {
2       bug();
3   }

```

Before execution starts DSE initializes the worklist with a state pointing to the start of the program (Line 12): (1, *true*, {}). After it fetches the conditional branch instruction, DSE will have to fork two new states in `ExecuteInstruction`: (2, *input_char* = 'B', {}) for the taken branch and (3, *input_char* ≠ 'B', {}) for the non-taken. Generating a test case for each execution path is straightforward: we send each path predicate to an SMT solver and any satisfying assignment will execute the same path, for example, *input_char* → 'B' to reach the buggy line of code. An unsatisfiable path predicate means the selected path is infeasible.

Advantages/Disadvantages. Forking executors and analyzing a single path at a time has benefits: the analysis code is simple, solving the generated path predicates is typically fast (e.g., in SAGE⁴ 99% of all queries takes less than 1 s) since we only reason about a single path, and the concrete path-specific state resolves several practical problems. For example, executors can execute hard-to-model functionality concretely (e.g., system calls), side effects such as allocating memory in each DSE path are reasoned about independently without extra work, and loops are unrolled as the code executes. The disadvantage is path explosion: the number of executors can grow exponentially in the number of branches. The path explosion problem is the main motivation for our veritestng algorithm (see Section 3).

2.2. Static symbolic execution (SSE)

SSE is a verification technique for representing a program as a logical formula. Safety checks are encoded as logical assertions that will falsify the formula if safety is violated. Because SSE checks programs, not paths, it is typically employed to verify the absence of bugs. As we will see, veritestng repurposes SSE techniques for summarizing program fragments instead of verifying complete programs.

Modern SSE algorithms summarize the effects of both branches at path confluence points. In contrast, DSE traditionally forks off two executors at the same line, which remain subsequently forever independent. Due to space, we do not repeat complete SSE algorithms here, and refer the reader to previous work.^{3, 15, 23}

Advantages/Disadvantages. Unlike DSE, SSE does not suffer from path explosion. All paths are encoded in a

^a Note the solver may still have to reason internally about an exponential number of paths—finding a satisfying assignment to a logical formula is an NP-hard problem.

single formula that is then passed to the solver.^a For acyclic programs, existing techniques allow generating compact formulas of size $O(n^2)$,^{10, 18} where n is the number of program statements. Despite these advantages over DSE, state-of-the-art tools still have trouble scaling to very large programs.^{13, 16} Problems include the presence of loops (how many times should they be unrolled?), formula complexity (are the formulas solvable if we encode loops and recursion?), the absence of concrete state (what is the concrete environment the program is running in?), as well as unmodeled behavior (a kernel model is required to emulate system calls). Another hurdle is completeness: for the verifier to prove absence of bugs, *all* program paths must be checked.

3. VERITESTING

DSE has proven to be effective in analyzing real world programs.^{6, 12} However, the path explosion problem can severely reduce the effectiveness of the technique. For example, consider the following 7-line program that counts the occurrences of the character 'B' in an input string:

```
1 int counter = 0, values = 0;
2 for ( i = 0 ; i < 100 ; i ++ )
3     if (input[i] == 'B') {
4         counter ++;
5         values += 2;
6     }
7 if (counter == 75)    bug ( );
```

The program above has 2^{100} possible execution paths. Each path must be analyzed separately by DSE, thus making full path coverage unattainable for practical purposes. In contrast, two test cases suffice for obtaining full code coverage: a string of 75 'B's and a string with no 'B's. However, finding such test cases in the 2^{100} state space is challenging.^b We ran the above program with several state-of-the-art symbolic executors, including KLEE,⁶ S2E,⁸ Mayhem,⁷ and Cloud9 with state merging.¹⁶ None of the above systems was able to find the bug within a 1-h time limit (they ran out of memory or kept running). Veritesting allows us to find the bug and obtain full path coverage in 47 s on the same hardware.

Veritesting starts with DSE, but switches to an SSE-style approach when we encounter code that—similar to the example above—does not contain system calls, indirect jumps, or other statements that are difficult to precisely reason about statically. Once in SSE mode, veritesting performs analysis on a dynamically recovered control flow graph (CFG) and identifies a core of statements that are easy for SSE, and a frontier of hard-to-analyze statements. The SSE algorithm summarizes the effects of all paths through the easy nodes up to the hard frontier. Veritesting then switches back to DSE to handle the cases that are hard to treat statically.

In the rest of this section, we present the main algorithm and the details of the technique.

3.1. The algorithm

In default mode, MergePoint behaves as a typical dynamic symbolic executor. It starts exploration with a concrete seed and explores paths in the neighborhood of the original seed following a generational search strategy.¹² MergePoint does not always fork when it encounters a symbolic branch. Instead, MergePoint intercepts the forking process—as shown in Line 10 (highlighted in green) of algorithm 1—of DSE and performs veritesting.

Veritesting consists of four main steps:

1. *CFG Recovery*. Obtains the CFG reachable from the address of the symbolic branch (Section 3.2).
2. *Transition Point Identification & Unrolling*. Takes in a CFG, and outputs candidate transition points and a CFG_e , an acyclic CFG with edges annotated with the control flow conditions (Section 3.3). Transition points indicate CFG locations with hard-to-model constructs where DSE may continue.
3. *SSE*. Takes the acyclic CFG_e and current execution state, and uses SSE to build formulas that encompass all feasible paths in the CFG_e . The output is a mapping from CFG_e nodes to SSE states (Section 3.4).
4. *Switch to DSE*. Given the transition points and SSE states, returns the DSE executors to be forked (Section 3.5).

3.2. CFG recovery

The goal of the CFG recovery phase is to obtain a partial CFG of the program, where the entry point is the current symbolic branch. We now define the notion of underapproximate and overapproximate CFG recovery.

A recovered CFG is an underapproximation if all edges of the CFG represent feasible paths. A recovered CFG is an overapproximation if all feasible paths in the program are represented by edges in the CFG (statically recovering a perfect—that is, non-approximate—CFG on binary code can be non-trivial). A recovered CFG might be an underapproximation or an overapproximation, or even both in practice.

Veritesting was designed to handle both underapproximated and overapproximated CFGs without losing paths or precision (see Section 3.4). MergePoint uses a customized CFG recovery mechanism designed to stop recovery at function boundaries, system calls and unknown instructions.

The output of this step is a partial (possibly approximate) intraprocedural CFG. Unresolved jump targets (e.g., `ret`, `call`, etc.) are forwarded to a generic `Exit` node in the CFG. Figure 1a shows the form of an example CFG after the recovery phase.

3.3. Transition point identification and unrolling

Once the CFG is obtained, MergePoint proceeds to identifying a set of transition points. Transition points define the boundary of the SSE algorithm (where DSE will continue exploration). Note that every possible execution path from the entry of the CFG needs to end in a transition point (our

^b For example, $\binom{100}{75} \approx 2^{78}$ paths reach the buggy line of code. The probability of finding one of those paths by random selection is approximately $2^{78}/2^{100} = 2^{-22}$.

implementation uses domination analysis²).

For a fully recovered CFG, a single transition point may be sufficient, for example, the bottom node in Figure 1a. However, for CFGs with unresolved jumps or system calls, any predecessor of the Exit node will be a possible transition point (e.g., the ret node in Figure 1b). Transition points represent the frontier of the visible CFG, which stops at unresolved jumps, function boundaries and system calls. The number of transition points gives an upper-bound on the number of executors that may be forked.

Unrolling Loops. Loop unrolling represents a challenge for static verification tools. However, MergePoint is dynamic and can concretely execute the CFG to identify how many times each loop will execute. The number of concrete loop iterations determines the number of loop unrolls. MergePoint also allows the user to extend loops beyond the concrete iteration limit, by providing a minimum number of unrolls.

To make the CFG acyclic, back edges are removed and forwarded to a newly created node for each loop, for example, the “Incomplete Loop” node in Figure 1b, which is a new transition point that will be explored if executing the loop more times is feasible. In a final pass, the edges of the CFG are annotated with the conditions required to follow the edge.

The end result of this step is a CFG_e and a set of transition points. Figure 1b shows an example CFG—without edge conditions—after transition point identification and loop unrolling.

3.4. Static symbolic execution

Given the CFG_e, MergePoint applies SSE to summarize the execution of multiple paths. Previous work,³ first converted the program to Gated Single Assignment (GSA)²² and then performed symbolic execution. In MergePoint, we encode SSE as a single pass dataflow analysis where GSA is computed on the fly—more details can be found in the full paper.²

To illustrate the algorithm, we run SSE on the following program:

```
if (x > 1) y = 1; else if (x < 42) y = 17;
```

Figure 1. Veritesting on a program fragment with loops and system calls. (a) Recovered CFG. (b) CFG after transition point identification & loop unrolling. Unreachable nodes are shaded.

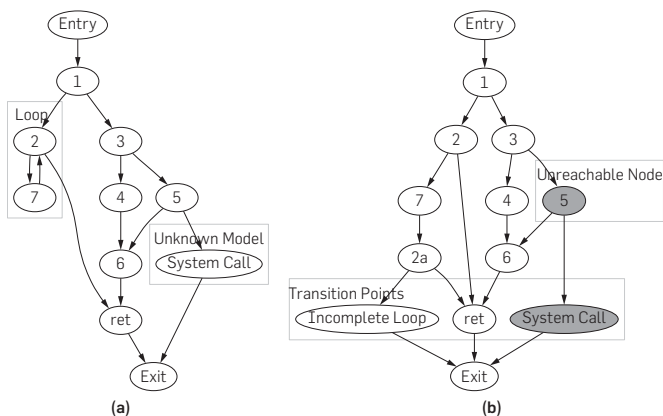


Figure 2 shows the progress of the variable state as SSE iterates through the blocks. SSE starts from the entry of the CFG_e and executes basic blocks in topological order. SSE uses conditional *ite* (if-then-else) expressions—*ite* is a ternary operator similar to `?:` in C—to encode the behavior of multiple paths. For example, every variable assignment following the true branch after the condition ($x > 1$) in Figure 2 will be guarded as $ite(x > 1, value, \perp)$, where *value* denotes the assigned value and $\perp$ is a don't care term. Thus, for the edge from B3 to B6 in Figure 2, Δ is updated to $\{y \rightarrow ite(x > 1, 42, \perp)\}$.

When distinct paths (with distinct Δ 's) merge to the same confluence point on the CFG, a merge operator is needed to “combine” the side effects from all incoming edges. To do so, we apply the following recursive merge operation $\mathcal{M}$ to each symbolic value:

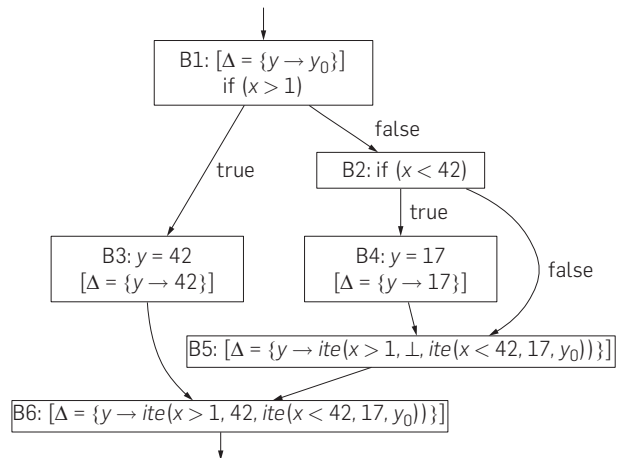
$$\begin{aligned} \mathcal{M}(v_1, \perp) &= v_1; & \mathcal{M}(\perp, v_2) &= v_2; \\ \mathcal{M}(ite(e, v_1, v_2), ite(e, v'_1, v'_2)) &= ite(e, \mathcal{M}(v_1, v'_1), \mathcal{M}(v_2, v'_2)) \end{aligned}$$

This way, at the last node of Figure 2, the value of *y* will be $\mathcal{M}(ite(x > 1, 42, \perp), ite(x > 1, \perp, ite(x < 42, 17, y_0)))$ which is merged to $ite(x > 1, 42, ite(x < 42, 17, y_0))$, capturing all possible paths.^c Note that this transformation is inlining multiple statements into a single one using *ite* operators. Also, note that values from unmerged paths ($\perp$ values) can be immediately simplified, for example, $ite(e, x, \perp) = x$. During SSE, MergePoint keeps a mapping from each traversed node to the corresponding variable state.

Handling Overapproximated CFGs. At any point during SSE, the path predicate is computed as the conjunction of the DSE predicate Π_{DSE} and the SSE predicate computed by substitution: Π_{SSE} . MergePoint uses the resulting predicate to perform path pruning offering two advantages: any infeasible edges introduced by CFG recovery are eliminated, and our formulas only consider feasible paths (e.g., the shaded

^c To efficiently handle deeply nested and potentially duplicated expressions, MergePoint utilizes hash-consing at the expression level.²

Figure 2. SSE running on an unrolled CFG—the variable state (Δ) is shown within brackets.



nodes in Figure 1b can be ignored).

3.5. Switch to DSE

After the SSE pass is complete, we check which states need to be forked. We first gather transition points and check whether they were reached by SSE. For the set of distinct, reachable transition points, MergePoint will fork a new symbolic state in a final step, where a DSE executor is created (pc, Π, Δ) using the state of each transition point.

Generating Test Cases. Though MergePoint can generate an input for each covered path, that would result in an exponential number of test cases in the size of the CFG. By default, we only output one test per CFG node explored by SSE. (Note that for branch coverage the algorithm can be modified to generate a test case for every edge of the CFG.) The number of test cases can alternatively be minimized by generating test cases only for nodes that have not been covered by previous test cases.

Underapproximated CFGs. Last, before proceeding with DSE, veritesting checks whether we missed any paths due to the underapproximated CFG. To do so, veritesting queries the negation of the path predicate at the `Exit` node (the disjunction of the path predicates of forked states). If the query is satisfiable, an extra state is forked to explore missed paths.

4. EVALUATION

In this section we evaluate our techniques using multiple benchmarks with respect to three main questions:

1. Does Veritesting find more bugs than previous approaches? We show that MergePoint with veritesting finds twice as many bugs than without.
2. Does Veritesting improve node coverage? We show MergePoint with veritesting improves node coverage over DSE.
3. Does Veritesting improve path coverage? Previous work showed dynamic state merging outperforms vanilla DSE.¹⁶ We show MergePoint with veritesting improves path coverage and outperforms both approaches.

We detail our large-scale experiment on 33,248 programs from Debian Linux. MergePoint generated billions of SMT queries, hundreds of millions of test cases, millions of crashes, and found 11,687 distinct bugs.

Overall, our results show MergePoint with veritesting improves performance on all three metrics. We also show that MergePoint is effective at checking a large number of programs. Before proceeding to the evaluation, we present our setup and benchmarks sets. All experimental data from MergePoint are publicly available online.²⁰

Experiment Setup. We ran all distributed MergePoint experiments on a private cluster consisting of 100 virtual nodes running Debian Squeeze on a single Intel 2.68GHz Xeon core with 1GB of RAM. All comparison tests against previous systems were run on a single node Intel Core i7 CPU and 16GB of RAM since these systems could not run on our distributed infrastructure.

We created three benchmarks: coreutils, BIN, and Debian. Coreutils and BIN were compiled so that coverage information could be collected via `gcov`. The Debian benchmark consists of binaries used by millions of users worldwide.

Benchmark 1: GNU coreutils (86 programs) We use the

coreutils benchmark to compare to previous work since: (1) the coreutils suite was originally used by KLEE⁶ and other researchers^{6, 7, 16} to evaluate their systems, and (2) configuration parameters for these programs used by other tools are publicly available.⁶ Numbers reported with respect to coreutils do not include library code to remain consistent with compared work. Unless otherwise specified, we ran each program in this suite for 1h.

Benchmark 2: The BIN suite (1023 programs). We obtained all the binaries located under the `/bin`, `/usr/bin`, and `/sbin` directories from a default Debian Squeeze installation.^d We kept binaries reading from `/dev/stdin`, or from a file specified on the command line. In a final processing step, we filtered out programs that require user interaction (e.g., GUIs). BIN consists of 1023 binary programs, and comprises 2,181,735 executable lines of source code (as reported by `gcov`). The BIN benchmark *includes* library code packaged with the application in the dataset, making coverage measurements more conservative than coreutils. For example, an application may include an entire library, but only one function is reachable from the application. We nonetheless include all uncovered lines from the library source file in our coverage computation. Unless otherwise specified, we ran each program in this suite for 30 min.

Benchmark 3: Debian (33,248 programs). This benchmark consists of all binaries from Debian Wheezy and Sid. We extracted binaries and shared libraries from every package available from the main Debian repository. We downloaded 23,944 binaries from Debian Wheezy, and 27,564 binaries from Debian Sid. After discarding duplicate binaries in the two distributions, we are left with a benchmark comprising 33,248 binaries. This represents an order of magnitude more applications than have been tested by prior symbolic execution research. We analyzed each application for less than 15 min per experiment.

4.1. Bug finding

Table 1 shows the number of bugs found by MergePoint with and without veritesting. Overall, veritesting finds 2× more bugs than without for BIN. Veritesting finds 63 (83%) of the bugs found without veritesting, as well as 85 additional distinct bugs that traditional DSE could not detect.

Veritesting also found two previously unknown crashes in coreutils, even though these applications have been thoroughly tested with symbolic execution.^{6, 7, 16} Further investigation showed that the coreutils crashes originate from a library bug that had been undetected for 9 years. The bug is in the time zone parser of the GNU portability library `Gnulib`, which dynamically deallocates a statically allocated memory buffer. It can be triggered by running `touch -d 'TZ=""' ,` or `date -d 'TZ=""' .` Furthermore, `Gnulib` is used by

^d What better source of benchmark programs than the ones you use everyday?

Table 1. Veritesting finds 2× more bugs.

	Veritesting	DSE
Coreutils	2 bugs/2 progs	0/0
BIN	148 bugs/69 progs	76 bugs/49 progs

several popular projects, and we have confirmed that the bug affects other programs, for example, `find`, `patch`, `tar`.

4.2. Node coverage

We evaluated MergePoint both with and without Veritest-ing on node coverage. Table 2 shows our overall results. Veritest-ing improves node coverage on average in all cases. Note that any positive increase in coverage is important. In particular, Kuznetsov et al. showed both dynamic state merging and SSE *reduced* node coverage when compared to vanilla DSE (Figure 8 in Ref.¹⁶).

Figures 3 and 4 break down the improvement per program. For coreutils, enabling veritest-ing decreased coverage in only three programs (`md5sum`, `printf`, and `pr`). Manual investigation of these programs showed that veritest-ing generated much harder formulas, and spent more than 90% of its time in the SMT solver, resulting in timeouts. In Figure 4 for BIN, we omit programs where node coverage was the same for readability. Overall, the BIN performance improved for 446 programs and decreased for 206.

Figure 5 shows the average coverage over time achieved by MergePoint with and without veritest-ing for the BIN suite. After 30 min, MergePoint without veritest-ing reached 34.45% code coverage. Veritest-ing achieved the same coverage in less than half the original time (12 min 48s). Veritest-ing's coverage improvement becomes more substantial as analysis time goes on. Veritest-ing achieved higher coverage

Table 2. Veritest-ing improves node coverage.

	Veritest-ing (%)	DSE (%)	Difference (%)
Coreutils	75.27	63.62	+11.65
BIN	40.02	34.71	+5.31

Figure 3. Code coverage difference on coreutils before and after veritest-ing.

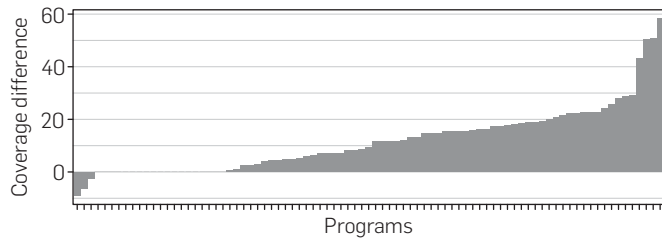
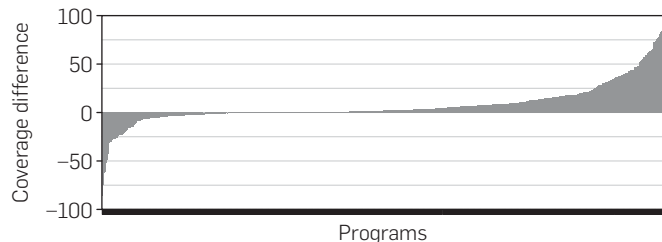


Figure 4. Code coverage difference on BIN before and after veritest-ing, where it made a difference.



velocity, that is, the rate at which new coverage is obtained, than standard symbolic execution. Over a longer period of time, the difference in velocity means that the coverage difference between the two techniques is likely to increase further, showing that the longer MergePoint runs, the more essential veritest-ing becomes for high code coverage.

The above tests demonstrates the improvements of veritest-ing for MergePoint. We also ran both S2E and MergePoint (with veritest-ing) on coreutils using the same configuration for 1 h on each utility in coreutils, excluding 11 programs where S2E emits assertion errors. Figure 6 compares the increase in coverage obtained by MergePoint with veritest-ing over S2E. MergePoint achieved 27% more code coverage on average than S2E. We investigated programs where S2E outperforms MergePoint. For instance, on `pinky`—the main outlier in the distribution—S2E achieves 50% more coverage. The main reason for this difference is that `pinky` uses a system call not handled by the current MergePoint implementation (netlink socket).

4.3. Path coverage

We evaluated the path coverage of MergePoint both with and without veritest-ing using three different metrics: time to complete exploration, as well as multiplicity.

Time to complete exploration. The metric reports the amount of time required to completely explore a program, in those cases where exploration finished.

The number of paths checked by an exhaustive DSE run is also the total number of paths possible. In such cases we can measure (a) whether veritest-ing also completed, and (b) if so, how long it took relative to DSE. MergePoint without veritest-ing was able to exhaust all paths for 46 programs. MergePoint with veritest-ing completes all paths 73% faster than without veritest-ing. This result shows that veritest-ing

Figure 5. Coverage over time (BIN suite).

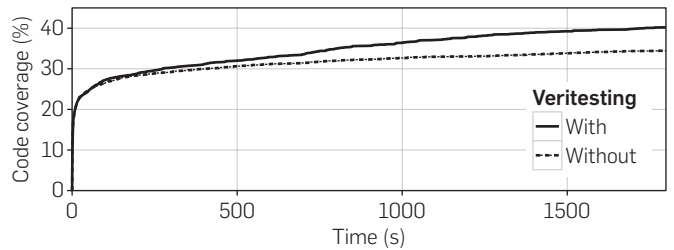
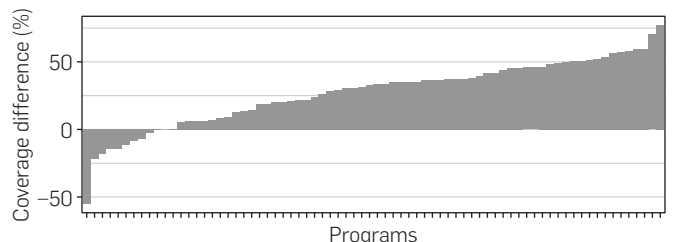


Figure 6. Code coverage difference on coreutils obtained by MergePoint versus S2E.



is faster when reaching the same end goal.

Multiplicity. Multiplicity was proposed by Kuznetsov et al.¹⁶ as a metric correlated with path coverage. The initial multiplicity of a state is 1. When a state forks, both children inherit the state multiplicity. When combining two states, the multiplicity of the resulting state is the sum of their multiplicities. A higher multiplicity indicates higher path coverage.

We also evaluated the multiplicity for veritesting. Figure 7 shows the state multiplicity probability distribution function for BIN. The average multiplicity over all programs was 1.4×10^{290} and the median was 1.8×10^{12} (recall, higher is better). The distribution resembles a lognormal with a spike for programs with multiplicity of 4096 (2^{12}). The multiplicity average and median for coreutils were 1.4×10^{199} and 4.4×10^{11} , respectively. Multiplicity had high variance; thus the median is likely a better performance estimator.

4.4. Checking Debian

In this section, we evaluate veritesting's bug finding ability on every program available in Debian Wheezy and Sid. We show that veritesting enables large-scale bug finding.

Since we test 33,248 binaries, any type of per-program manual labor is impractical. We used a single input specification for our experiments: `-sym-arg 1 10 -sym-arg 2 2 -sym-arg 3 2 -sym-anon-file 24 -sym-stdin 24` (3 symbolic arguments up to 10, 2, and 2 bytes, respectively, and symbolic files/stdin up to 24 bytes). MergePoint encountered at least one symbolic branch in 23,731 binaries. We analyzed Wheezy binaries once, and Sid binaries twice (one experiment with a 24-byte symbolic file, the other with 2100 bytes to find buffer overflows). Including data

Figure 7. Multiplicity distribution (BIN suite).

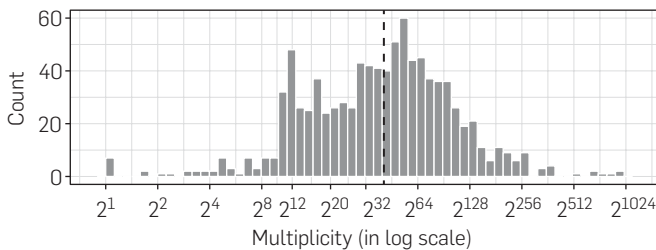


Table 3. Overall numbers for checking Debian.

Total programs	33,248
Total SMT queries	15,914,407,892
Queries hitting cache	12,307,311,404
Symbolic instrs	71,025,540,812
Run time	235,623,757 s
Symb exec time	125,412,247 s
SAT time	40,411,781 s
Model gen time	30,665,881 s
# test cases	199,685,594
# crashes	2,365,154
# unique bugs	11,687
# reported bugs	1043
# fixed bugs	229

processing, the experiments took 18 CPU-months.

Our overall results are shown in Table 3. Veritesting found 11,687 distinct bugs that crash programs. The bugs appear in 4379 of the 33,248 programs. Veritesting also finds bugs that are potential security threats. Two hundred and twenty-four crashes have a corrupt stack, that is, a saved instruction pointer has been overwritten by user input. As an interesting data point, it would have cost \$0.28 per unique crash had we run our experiments on the Amazon Elastic Compute Cloud, assuming that our cluster nodes are equivalent to large instances.

The volume of bugs makes it difficult to report all bugs in a usable manner. Note that each bug report includes a crashing test case, thus reproducing the bug is easy. Instead, practical problems such as identifying the correct developer and ensuring responsible disclosure of potential vulnerabilities dominate our time. As of this writing, we have reported 1043 crashes in total.¹⁹ Not a single report was marked as unreproducible on the Debian bug tracking system. Two hundred and twenty-nine bugs have already been fixed in the Debian repositories, demonstrating the real-world impact of our work. Additionally, the patches gave an opportunity to the package maintainers to harden at least 29 programs, enabling modern defenses like stack canaries and DEP.

4.5. Discussion

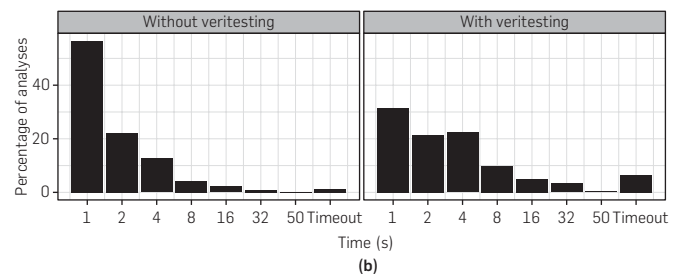
Our experiments so far show that veritesting can effectively increase multiplicity, achieve higher code coverage, and find more bugs. In this section, we discuss why it works well according to our collected data.

Each run takes longer with veritesting because multi-path SMT formulas tend to be harder. The coverage improvement demonstrates that additional SMT cost is amortized over the increased number of paths represented in each run. At its core, veritesting is pushing the SMT engine harder instead of brute-forcing paths by forking new DSE executors. This result confirms that the benefits of veritesting outweigh its cost. The distribution of path times (Figure 8b) shows that the vast majority (56%) of paths explored take less than 1 s for standard symbolic execution. With veritesting, the fast paths are fewer (21%), and we get more timeouts (6.4% vs.

Figure 8. MergePoint performance before and after veritesting for BIN. The above figures show: (a) Performance breakdown for each component; (b) Analysis time distribution.

Component	DSE (%)	Veritesting (%)
Instrumentation	40.01	16.95
SMT solver	19.23	63.16
Symbolic execution	39.76	19.89

(a)



1.2%). The same differences are also reflected in the component breakdown. With veritesting, most of the time (63%) is spent in the solver, while with standard DSE most of the time (60%) is spent re-executing similar paths that could be merged and explored in a single execution.

Of course there is no free lunch, and some programs do perform worse. We emphasize that on average over a fairly large dataset our results indicate the tradeoff is beneficial.

5. RELATED WORK

Symbolic execution was discovered in 1975,¹⁴ with the volume of academic research and commercial systems exploding in the last decade. Notable symbolic executors include SAGE and KLEE. SAGE⁴ is responsible for finding one third of all bugs discovered by file fuzzing during the development of Windows 7.⁴ KLEE⁶ was the first tool to show that symbolic execution can generate test cases that achieve high coverage on real programs by demonstrating it on the UNIX utilities. There is a multitude of symbolic execution systems—for more details, we refer the reader to recent surveys.^{5,21}

Merging execution paths is not new. Koelbl and Pixley¹⁵ pioneered path merging in SSE. Concurrently and independently, Xie and Aiken²³ developed Saturn, a verification tool capable of encoding of multiple paths before converting the problem to SAT. Hansen et al.¹³ follow an approach similar to Koelbl et al. at the binary level. Babic and Hu³ improved their static algorithm to produce smaller and faster to solve formulas by leveraging GSA.²² The static portion of our veritesting algorithm is built on top of their ideas. In our approach, we alternate between SSE and DSE. Our approach amplifies the effect of DSE and takes advantage of the strengths of both techniques.

The efficiency of the static algorithms mentioned above typically stems from various types of *if-conversion*,¹ a technique for converting code with branches into predicated straightline statements. The technique is also known as ϕ -folding,¹⁷ a compiler optimization technique that collapses simple diamond-shaped structures in the CFG. Godefroid¹¹ introduced function summaries to test code compositionally. The main idea is to record the output of an analyzed function, and reuse it whenever the function is called again. Veritesting generates context-sensitive on-demand summaries of code fragments as the program executes—extending to compositional summaries is possible future work.

6. CONCLUSION

In this article we proposed MergePoint and veritesting, a new technique to enhance symbolic execution with verification-based algorithms. We evaluated MergePoint on 1023 programs and showed that veritesting increases the number of bugs found, node coverage, and path coverage. We showed that veritesting enables large-scale bug finding by testing 33,248 Debian binaries, and finding 11,687 bugs. Our results have had real world impact with 229 bug fixes already present in the latest version of Debian.

Acknowledgments

We would like to thank Samantha Gottlieb, Tiffany Bao, and our anonymous reviewers for their comments and suggestions. We also thank Mitch Franzos and PDL for the support

they provided during our experiments. This research was supported in part by grants from DARPA and the NSF, as well as the Prabhu and Poonam Goel Fellowship. 

References

- Allen, J.R., Kennedy, K., Porterfield, C., Warren, J. Conversion of control dependence to data dependence. In *Proceedings of the 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages* (Austin, Texas, 1983). ACM Press, New York, NY, 177–189.
- Avgerinos, T., Rebert, A., Cha, S.K., Brumley, D. Enhancing symbolic execution with veritesting. In *Proceedings of the 36th International Conference on Software Engineering, ICSE 2014* (Hyderabad, India, 2014). ACM, New York, NY, 1083–1094. DOI: 10.1145/2568225.2568293. URL <http://doi.acm.org/10.1145/2568225.2568293>.
- Babic, D., Hu, A.J. Calysto: Scalable and precise extended static checking. In *Proceedings of the 30th International Conference on Software Engineering* (Leipzig, Germany, 2008). ACM, New York, NY, 211–220.
- Bounimova, E., Godefroid, P., Molnar, D. Billions and billions of constraints: Whitebox Fuzz testing in production. In *Proceedings of the 35th IEEE International Conference on Software Engineering* (San Francisco, CA, 2013). IEEE Press, Piscataway, NJ, 122–131.
- Cadar, C., Sen, K. Symbolic execution for software testing: three decades later. *Commun. ACM* 56, 2 (2013), 82–90.
- Cadar, C., Dunbar, D., Engler, D. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the 8th USENIX Symposium on Operating System Design and Implementation* (San Diego, CA, 2008). USENIX Association, Berkeley, CA, 209–224.
- Cha, S.K., Avgerinos, T., Rebert, A., Brumley, D. Unleashing mayhem on binary code. In *Proceedings of the 33rd IEEE Symposium on Security and Privacy* (2012). IEEE Computer Society, Washington, DC, 380–394.
- Chipounov, V., Kuznetsov, V., Candea, G. S2E: A platform for in vivo multi-path analysis of software systems. In *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems* (Newport Beach, CA, 2011). ACM, New York, NY, 265–278.
- de Moura, L., Bjørner, N. Satisfiability modulo theories: Introduction and applications. *Commun. ACM* 54, 9 (Sept. 2011), 69. ISSN 00010782. doi: 10.1145/1995376.1995394. URL <http://dl.acm.org/citation.cfm?doid=1995376.1995394>.
- Flanagan, C., Saxe, J. Avoiding exponential explosion: Generating compact verification conditions. In *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (London, United Kingdom, 2001). ACM, New York, NY, 193–205.
- Godefroid, P. Compositional dynamic test generation. In *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Nice, France, 2007). ACM, New York, NY, 47–54.
- Godefroid, P., Levin, M.Y., Molnar, D. SAGE: Whitebox fuzzing for security testing. *Commun. ACM* 55, 3 (2012), 40–44.
- Hansen, T., Schachte, P., Søndergaard, H. State joining and splitting for the symbolic execution of binaries. *Runtime Verif.* (2009), 76–92.
- King, J.C. Symbolic execution and program testing. *Commun. ACM* 19, 7 (1976), 385–394.
- Koelbl, A., Pixley, C. Constructing efficient formal models from high-level descriptions using symbolic simulation. *Int. J. Parallel Program.* 33, 6 (Dec. 2005), 645–666.
- Kuznetsov, V., Kinder, J., Bucur, S., Candea, G. Efficient state merging in symbolic execution. In *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation* (Beijing, China, 2012). ACM, New York, NY, 193–204.
- Lattner, C., Adve, V. LLVM: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization* (Palo Alto, CA, 2004). IEEE Computer Society, Washington, DC, 75–86.
- Leino, K.R.M. Efficient weakest preconditions. *Inform. Process. Lett.* 93, 6 (2005), 281–288.
- Mayhem. 1.2K Crashes in Debian, 2013. URL <http://lists.debian.org/debian-devel/2013/06/msg00720.html>.
- Mayhem. Open Source Statistics & Analysis, 2013. URL <http://www.forallsecure.com/summaries>.
- Schwartz, E.J., Avgerinos, T., Brumley, D. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *Proceedings of the 31st IEEE Symposium on Security and Privacy* (2010). IEEE Computer Society, Washington, DC, 317–331.
- Tu, P., Padua, D. Efficient building and placing of gating functions. In *Proceedings of the 16th ACM Conference on Programming Language Design and Implementation* (La Jolla, CA, 1995). ACM, New York, NY, 47–55.
- Xie, Y., Aiken, A. Scalable error detection using boolean satisfiability. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Long Beach, CA, 2005). ACM, New York, NY, 351–363.

Thanassis Avgerinos, Alexandre Rebert, and David Brumley ([thanassis, alex]@forallsecure.com), For AllSecure, Inc., Pittsburgh, PA.

Thanassis Avgerinos, Alexandre Rebert, Sang Kil Cha, and David Brumley ([sangkilc, dbrumley]@cmu.edu), Carnegie Mellon University, Pittsburgh, PA.

Technical Perspective

Computing with the Crowd

By Siddharth Suri

COMPUTER SCIENCE IS primarily focused on computation using microprocessors or CPUs. However, the recent rise in the popularity of crowdsourcing platforms, like Amazon's Mechanical Turk, provides another computational device—the crowd. Crowdsourcing is the act of outsourcing a job to an undefined group of people, known as the crowd, through an open call.³ Crowdsourcing platforms are online labor markets where employers can post jobs and workers can do jobs for pay, but they can also be viewed as distributed computational systems where the workers are the CPUs and will perform computations for pay. In other words, crowdsourcing platforms provide a way to execute computation with humans. In a traditional computational system when a programmer wants to compute something, they interact with a CPU through an API defined by an operating system. But in a crowdsourcing environment, when a programmer wants to compute something, they interact with a human through an API defined by a crowdsourcing platform.

Why might one want to do computation with humans? There are a variety of problems that are easy for humans but difficult for machines. Humans have pattern-matching skills and linguistic-recognition skills that machines have been unable to match as of yet. For example, FoldIt¹ is a system where people search for the natural configuration of proteins and their results often outperform solutions computed using only machines. Conversely, there are problems that are easy for machines to solve but difficult for humans. Machines excel at computation on massive datasets since they can do the same operations repeatedly without getting tired or hungry. This brings up the natural question: What kinds of problems can be solved with both human and machine computation that neither could do alone?

Systems like AutoMan, described in the following paper by Barowy et al., provide the first steps toward answering this question. AutoMan is a domain-specific programming language that provides an abstraction layer on top of the crowd. It allows the programmer to interleave the expression of computation using both humans and machines in the same program. In an AutoMan program, one function could be executed by a CPU and the next could be executed by humans.

This new type of computation brings new types of complexity, which AutoMan is designed to manage. Most of this complexity stems from the fact that unlike CPUs, humans have agency. They make decisions; they have needs, wants, and biases. Humans can choose what tasks to do, when to quit, what is and isn't worth their time, and when to communicate with another human and what about. CPUs, on the other hand, always execute whatever instructions they are given. Much of the design and implementation of AutoMan addresses this key difference between humans and machines. For example, AutoMan has extensive functionality for quality control on the output of the workers. It also has functionality to discover the price that will be enough to incentivize workers to do the given task and to reduce collusion among workers. Computation with CPUs does not require any of this functionality. AutoMan also addresses the natural difference in speed between human and machine computation by allowing eager evaluation of the machine commands and only blocking on the humans when necessary.

Being able to express human computation and interleave human and machine computation opens up interesting new research directions in human computation and organizational dynamics. In the nascent field

of human computation, since we can now express human computation in a programming language, we can next develop a model of human computation analogous to the PRAM.² This would, in turn, allow us to develop a theory of complexity for human computation to help us understand what problems are easy and difficult for humans to solve. Developing these theories might help us scale up AutoMan, which is currently designed to solve microtasks, in terms of complexity to solve bigger tasks and workflows.

Taking a broader and more interdisciplinary perspective, one can view a company as a computational device that combines the human computation of its employees with the machine computation of the company's computers. A better theoretical and empirical understanding of human computation could allow the field of computer science to inform how best to architect and organize companies for greater accuracy and efficiency. Whether or not AutoMan proves revolutionary as a programming language, it is important as an idea because it provides a “computational lens”⁴ on the science of crowdsourcing, human computation, and the study of group problem solving. 

References

1. Cooper, S. et al. Predicting protein structures with a multiplayer online game. *Nature* 446 (Aug. 2010), 756–760.
2. Fortune, S. and Wylie, J. Parallelism in random access machines. In *Proceedings of the 10th Annual Symposium on Theory of Computing* (1978). ACM, 114–118.
3. Howe, J. The rise of crowdsourcing. *Wired* (June 1, 2006).
4. Karp R.M. Understanding science through the computational lens. *J. Computer Science and Technology* 26, 4 (July 2011), 569–577.

Siddharth Suri (suri@microsoft.com) is Senior Researcher at—and one of the founding members of—Microsoft Research in New York City.

Copyright held by author.

AutoMan: A Platform for Integrating Human-Based and Digital Computation

By Daniel W. Barowy, Charlie Curtsinger, Emery D. Berger, and Andrew McGregor

Abstract

Humans can perform many tasks with ease that remain difficult or impossible for computers. Crowdsourcing platforms like Amazon Mechanical Turk make it possible to harness human-based computational power at an unprecedented scale, but their utility as a general-purpose computational platform remains limited. The lack of complete automation makes it difficult to orchestrate complex or interrelated tasks. Recruiting more human workers to reduce latency costs real money, and jobs must be monitored and rescheduled when workers fail to complete their tasks. Furthermore, it is often difficult to predict the length of time and payment that should be budgeted for a given task. Finally, the results of human-based computations are not necessarily reliable, both because human skills and accuracy vary widely, and because workers have a financial incentive to minimize their effort.

We introduce AUTOMAN, the first fully automatic crowd-programming system. AUTOMAN integrates human-based computations into a standard programming language as ordinary function calls that can be intermixed freely with traditional functions. This abstraction lets AUTOMAN programmers focus on their programming logic. An AUTOMAN program specifies a confidence level for the overall computation and a budget. The AUTOMAN runtime system then transparently manages all details necessary for scheduling, pricing, and quality control. AUTOMAN automatically schedules human tasks for each computation until it achieves the desired confidence level; monitors, reprices, and restarts human tasks as necessary; and maximizes parallelism across human workers while staying under budget.

1. INTRODUCTION

Humans perform many tasks with ease that remain difficult or impossible for computers. For example, humans are far better than computers at performing tasks like vision, motion planning, and natural language understanding.^{16, 18} Many researchers expect these “AI-complete” tasks to remain beyond the reach of computers for the foreseeable future.¹⁹ Harnessing human-based computation in general and at scale faces the following challenges:

Determination of pay and time for tasks. Employers must decide the payment and time allotted before posting tasks. It is both difficult and important to choose these correctly since workers will not accept tasks with too-short deadlines or too little pay.

Scheduling complexities. Employers must manage the tradeoff between latency (humans are relatively slow) and

cost (more workers means more money). Because workers may fail to complete their tasks in the allotted time, jobs need to be tracked and reposted as necessary.

Low quality responses. Human-based computations always need to be checked: worker skills and accuracy vary widely, and they have a financial incentive to minimize their effort. Manual checking does not scale, and majority voting is neither necessary nor sufficient. In some cases, majority vote is too conservative, and in other cases, it is likely that workers will agree by chance.

Contributions

We introduce AUTOMAN, a programming system that integrates human-based and digital computation. AUTOMAN addresses the challenges of harnessing human-based computation at scale:

Transparent integration. AUTOMAN abstracts human-based computation as ordinary function calls, freeing the programmer from scheduling, budgeting, and quality control concerns (Section 3).

Automatic scheduling and budgeting. The AUTOMAN runtime system schedules tasks to maximize parallelism across human workers while staying under budget. AUTOMAN tracks job progress, reschedules, and reprices failed tasks as necessary (Section 4).

Automatic quality control. The AUTOMAN runtime system manages quality control automatically. AUTOMAN creates enough human tasks for each computation to achieve the confidence level specified by the programmer (Section 5).

2. BACKGROUND

Since crowdsourcing is a novel application domain for programming language research, we summarize the necessary background on crowdsourcing platforms. We focus on Amazon Mechanical Turk^a (MTurk), but other crowdsourcing platforms are similar. MTurk acts as an intermediary between *requesters* and *workers* for short-term tasks.

Human intelligence task. In MTurk parlance, tasks are known as *human intelligence tasks* (HITs). Each HIT is

^a Amazon Mechanical Turk is hosted at <http://mturk.com>.

The original version of this paper was published in the *Proceedings of OOPSLA 2012*.

represented as a question form, composed of any number of questions and associated metadata such as a title, description, and search keywords. Questions can be either free-text questions, where workers provide a free-form textual response, or multiple-choice questions, where workers make one or more selections from a set of options. Most HITs on MTurk are for relatively simple tasks, such as “does this image match this product?” Compensation is generally low (usually a few cents) since employers expect that work to be completed quickly (on the order of seconds).

Requesting work. Requesters can create HITs using either MTurk’s website or programmatically, using an API. Specifying a number of *assignments* greater than one allows multiple unique workers to complete the same task, parallelizing HITs. Distinct HITs with similar qualities can also be grouped to make it easy for workers to find similar work.

Performing work. Workers may choose any available task, subject to qualification requirements (see below). When a worker selects a HIT, she is granted a time-limited *reservation* for that particular piece of work such that no other worker can accept it.

HIT expiration. HITs have two timeout parameters: the amount of time that a HIT remains visible on MTurk, known as the *lifetime* of a HIT, and the amount of time that a worker has to complete an assignment once it is granted, known as the *duration* of an assignment. If a worker exceeds the assignment’s duration without submitting completed work, the reservation is cancelled, and the HIT becomes available to other workers. If a HIT reaches the end of its lifetime without its assignments having been completed, the HIT expires and is made unavailable.

Requesters: Accepting or rejecting work. Once a worker submits a completed assignment, the requester may then accept or reject the completed work. Acceptance indicates that the completed work is satisfactory, at which point the worker is paid. Rejection withholds payment. The requester may provide a textual justification for the rejection.

Worker quality. The key challenge in automating work in MTurk is attracting good workers and discouraging bad workers from participating. MTurk provides no mechanism for requesters to seek out specific workers (aside from emails). Instead, MTurk provides a *qualification* mechanism that limits which workers may participate. A common qualification is that workers must have an overall assignment-acceptance rate of 90%.

Given the wide variation in tasks on MTurk, overall worker accuracy is of limited utility. For example, a worker may be skilled at audio transcription tasks and thus have a high accuracy rating, but it would be a mistake to assume on the basis of their rating that the same worker could also perform Chinese-to-English translation tasks. Worse, workers who cherry-pick easy tasks and thus have high accuracy ratings may be less qualified than workers who routinely perform difficult tasks that are occasionally rejected.

3. OVERVIEW

AUTOMAN is a domain-specific language embedded in Scala. AUTOMAN’s goal is to abstract away the details of crowdsourcing so that human computation can be as easy to invoke as a conventional programming language function.

3.1. Using AutoMan

Figure 1 presents a real AUTOMAN program that recognizes automobile license plate texts from images. Note that the programmer need not specify details about the chosen crowdsourcing backend (Mechanical Turk) other than the appropriate backend adapter and account credentials. Crucially, all details of crowdsourcing are hidden from the AUTOMAN programmer. The AUTOMAN runtime abstracts away platform-specific interoperability code, schedules and determines budgets (both cost and time), and automatically ensures that outcomes meet a minimum confidence level.

Initializing AutoMan. After importing the AUTOMAN and MTurk adapter libraries, the first thing an AUTOMAN programmer does is to declare a configuration for the desired crowdsourcing platform. The configuration is then bound to an AUTOMAN runtime object that instantiates any platform-specific objects.

Specifying AutoMan functions. AUTOMAN functions declaratively describe questions that workers must answer. They must include the question type and may also include text or images.

Confidence level. An AUTOMAN programmer can optionally specify the degree of confidence they want to have in their computation, on a per-function basis. AUTOMAN’s default confidence is 95%, but this can be overridden as needed. The meaning and derivation of confidence is discussed in Section 5.

Metadata and question text. Each question declaration requires a title and description, used by the crowdsourcing

Figure 1. A license plate recognition program written using AutoMan. `getURLsFromDisk()` is omitted for clarity. The AutoMan programmer specifies only credentials for Mechanical Turk, an overall budget, and the question itself; the AutoMan runtime manages all other details of execution (scheduling, budgeting, and quality control).

```
import edu.umass.cs.automan.adapters.MTurk._

object ALPR extends App {
  val a = MTurkAdapter { mt =>
    mt.access_key_id = "XXXXX"
    mt.secret_access_key = "XXXXX"
  }

  def plateTxt(url:String) = a.FreeTextQuestion { q =>
    q.budget = 5.00
    q.text = "What does this license plate say?"
    q.image_url = url
    q.allow_empty_pattern = true
    q.pattern = "XXXXXXXXXX"
  }

  automan(a) {
    // get plate texts from image URLs
    val urls = getURLsFromDisk()
    val plate_texts = urls.map { url =>
      (url, plateTxt(url))
    }

    // print out results
    plate_texts.foreach { (url,outcome) =>
      outcome.answer match {
        case Answer(ans, _) =>
          println(url + ": " + ans)
        case _ => ()
      }
    }
  }
}
```

platform's user interface. These fields map to MTurk's fields of the same name. A declaration also includes the question text itself, together with a map between symbolic constants and strings for possible answers.

Question variants. AUTOMAN supports multiple-choice questions, including questions where only one answer is correct ("radio-button" questions), where any number of answers may be correct ("checkbox" questions), and a restricted form of free-text entry. Section 5 describes how AUTOMAN's quality control algorithm handles each question type.

Invoking a function. A programmer can invoke an AUTOMAN function as if it were any ordinary (digital) function. In Figure 1, the programmer calls the `plateText` function with a URL pointing to an image as a parameter. The function returns an `Outcome` object representing a `Future[Answer]` that can then be passed as data to other functions. AUTOMAN functions execute eagerly, in a background thread, as soon as they are invoked. The program does not block until it needs to read an `Outcome.answer` field, and only then if the human computation is not yet finished.

4. SCHEDULING ALGORITHM

AUTOMAN's scheduler controls task marshaling, budgeting of time and cost, and quality. This section describes how AUTOMAN automatically determines these parameters.

4.1. Calculating timeout and reward

AUTOMAN's overriding goal is to recruit workers quickly and at low cost in order to keep the cost of a computation within the programmer's budget. AUTOMAN posts tasks in *rounds* that have a fixed timeout during which tasks must be completed. When AUTOMAN fails to recruit workers in a round, there are two possible causes: workers were not willing to complete the task for the given reward, or the time allotted was not sufficient. AUTOMAN does not distinguish between these cases. Instead, the reward for a task and the time allotted are both increased by a constant factor g every time a task goes unanswered. g must be chosen carefully to ensure the following two properties:

1. The reward for a task should quickly reach a worker's minimum acceptable compensation.
2. The reward should not grow so quickly that it incentivizes workers to wait for a larger reward.

Section 4.4 presents an analysis of reward growth rates. We also discuss the feasibility of our assumptions and possible attack scenarios in Section 5.4.

4.2. Scheduling the right number of tasks

AUTOMAN's default policy for spawning tasks is optimistic: it creates the smallest number of tasks required to reach the desired confidence level when workers agree unanimously. If workers do agree unanimously, AUTOMAN returns their answer. Otherwise, AUTOMAN computes and then schedules the minimum number of additional votes required to reach confidence.

When the user-specified budget is insufficient, AUTOMAN suspends the computation before posting additional tasks. The computation can either be resumed with an increased

budget or accepted as-is, with a confidence value lower than the one requested. The latter case is considered exceptional, and must be explicitly handled by the programmer.

4.3. Trading off latency and money

AUTOMAN allows programmers to provide a *time-value* parameter that counterbalances the default optimistic assumption that all workers will agree. The parameter instructs the system to post more than the minimum number of tasks in order to minimize the latency incurred when jobs are serialized across multiple rounds. The number of tasks posted is a function of the *value of the programmer's time*:

$$\max \left(\min_required, \left\lceil \frac{\text{time_value}}{\min_wage} \right\rceil \right).$$

As a cost savings, when AUTOMAN receives enough answers to reach the specified confidence, it cancels all unaccepted tasks. In the worst case, all posted tasks will be answered before AUTOMAN can cancel them, which will cost no more than $\text{time_value} \cdot \text{task_timeout}$. While this strategy runs the risk of paying substantially more for a computation, it can yield dramatic reductions in latency. We re-ran the example program described in Section 7.1 with a time-value set to \$50. In two separate runs, the computation completed in 68 and 168 *seconds*; by contrast, the default time-value (minimum wage) took between 1 and 3 *hours* to complete.

4.4. Maximum reward growth rate

When workers encounter a task with an initial reward of R they may choose to accept the task or wait for the reward to grow. If R is below $R_{\min}$, the smallest reward acceptable to workers, then tasks will not be completed. Let g be the reward growth rate and let i be the number of discrete time steps, or *rounds*, that elapse from an initial time $i = 0$, such that a task's reward after i rounds is $g^i R$. We want a g large enough to reach $R_{\min}$ quickly, but not so large that workers have an incentive to wait. We balance the probability that a task remains available against the reward's growth rate so workers should not expect to profit by waiting.

Let p_a be the probability that a task remains available from one round to the next, assuming this probability is constant across rounds. Suppose a worker's strategy is to wait i rounds and then complete the task for a larger reward. The expected reward for this worker's strategy is

$$\mathbb{E} [\text{reward}_i] = (p_a g)^i R.$$

when $g \leq 1/p_a$, the expected reward is maximized at $i = 0$; workers have no incentive to wait, even if they are aware of AUTOMAN's pricing strategy. A growth rate of exactly $1/p_a$ will reach $R_{\min}$ as fast as possible without incentivizing waiting. This pricing strategy remains sound even when p_a is not constant, provided the desirability of a task does not decrease with a larger reward.

The true value of p_a is unknown, but it can be estimated by modeling the acceptance or rejection of each task as an independent Bernoulli trial. The maximum likelihood estimator

$$\tilde{p}_a = \operatorname{argmax}_{p \in [0,1]} p^t (1-p)^{n-t} = \frac{t}{n}$$

is a reasonable estimate for p_a , where n is the number of times a task has been offered and t is the number of times the task was not accepted before timing out. To be conservative, $\tilde{p}_a$ can be over-approximated, driving g downward. The difficulty of choosing a reward a priori is a strong case for automatic budgeting.

5. QUALITY CONTROL

AUTOMAN's quality control algorithm is based on collecting enough consensus for a given question to rule out the possibility, for a specified level of confidence, that the results are due to random chance. AUTOMAN's algorithm is adaptive, taking both the programmer's confidence threshold and the likelihood of random agreement into account. By contrast, majority rule, a commonly used technique for achieving higher-quality results, is neither necessary nor sufficient to rule out outcomes due to random chance (see Figure 2). A simple two-option question (e.g., "Heads or tails?") with three *random* respondents demonstrates the problem: a majority is not just likely, it is guaranteed. Section 5.4 justifies this approach.

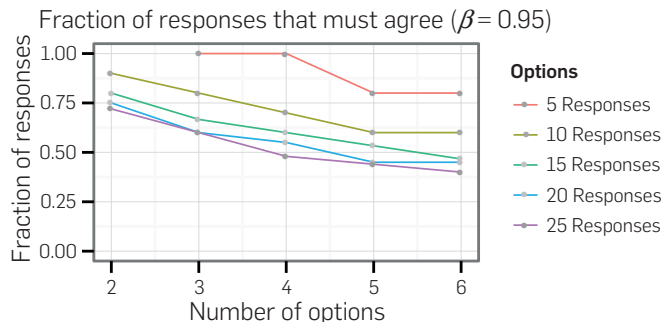
Initially, AUTOMAN spawns enough tasks to meet the desired confidence level if all workers who complete the tasks agree unanimously. Computing the confidence of an outcome in this scenario is straightforward. Let k be the number of options, and n be the number of tasks. The confidence is then $1 - k(1/k)^n$. AUTOMAN computes the smallest n such that the probability of random agreement is less than or equal to one minus the specified confidence threshold.

Humans are capable of answering a rich variety of question types. Each of these question types requires its own probability analysis.

○ **Radio** ● **Buttons** For multiple-choice "radio-button" questions where only one choice is possible, k is exactly the number of possible options.

☑ **Check** ☐ **Boxes** For "checkbox" questions with c boxes, k is much larger: $k = 2^c$. In practice, k is often large enough

Figure 2. The fraction of workers that must agree to reach 0.95 confidence for a given number of tasks. For a three-option question and 5 workers, 100% of the workers must agree. For a six-option question and 15 or more workers, only a plurality is required to reach confidence. Notice that majority vote is neither necessary nor sufficient to rule out random respondents.



that as few as two workers are required to rule out random behavior. To avoid accidental agreement caused when low-effort workers simply submit a form without changing any of the checkboxes, AUTOMAN randomly pre-selects checkboxes.

Free-text Input Restricted "Free-text" input is mathematically equivalent to a set of radio-buttons where each option corresponds to a valid input string. Nonetheless, even a small set of valid strings represented as radio buttons would be burdensome for workers. Instead, workers are provided with a text entry field and the programmer supplies a pattern representing valid inputs so that AUTOMAN can perform its probability analysis.

AUTOMAN's pattern specification syntax resembles COBOL's picture clauses. A matches an alphabetic character, B matches an optional alphabetic character, X matches an alphanumeric character, Y matches an optional alphanumeric character, 9 matches a numeric character, and 0 matches an optional numeric character. For example, a telephone number recognition application might use the pattern 09999999999.

For example, given a 7-character numeric pattern with no optional characters, $k = 10^7$. Again, k is often large, so a small number of HITs suffice to achieve high confidence in the result. As with checkbox questions, AUTOMAN treats free-text questions specially to cope with low-effort workers who might simply submit an empty string. To avoid this problem, AUTOMAN only accepts the empty string if it is explicitly entered with the special string NA.

5.1. Definitions

Formally, AUTOMAN's quality control algorithm depends on two functions, t and v , and associated parameters β and p^* . t computes the minimum threshold (the number of votes) needed to establish that an option is unlikely to be due to random chance with probability β (the programmer's confidence threshold). t depends on the random variable X , which models when n respondents choose one of k options uniformly at random. If no option crosses the threshold, v computes the additional number of votes needed. v depends on the random variable Y , which models a worker choosing the correct option with the observed probability p^* and all other options uniformly at random.

Let X and Y be multinomial distributions with parameters $(n, 1/k, \dots, 1/k)$ and $(n, p, q, \dots, q)$, respectively, where $q = (1-p)/(k-1)$. We define two functions $\mathcal{E}_1$ and $\mathcal{E}_2$ that have the following properties²:

LEMMA 5.1.

$$\Pr \left[\max_i X_i < m \right] = \mathcal{E}_1(n, m),$$

$$\Pr \left[\max_{i \geq 2} Y_i < m \leq Y_1 \right] = \mathcal{E}_2(p, n, m),$$

where

$$\mathcal{E}_1(n, m) = \frac{n!}{k^n} \cdot \text{coeff}_{\lambda, n} \left(\sum_{j=0}^{m-1} \frac{\lambda^j}{j!} \right)^k$$

and

$$\mathcal{E}_2(p, n, m) = n! \cdot \text{coeff}_{\lambda, n} \left(\sum_{j=0}^{m-1} \frac{(q\lambda)^j}{j!} \right)^{k-1} \cdot \sum_{j=m}^{\infty} \frac{(p\lambda)^j}{j!},$$

where $\text{coeff}_{\lambda, n}(f(\lambda))$ is the coefficient of λ^n in the polynomial f . Note that $\mathcal{E}_1(n, n) = 1 - 1/k^{n-1}$ and define

$$t(n, \beta) := \begin{cases} \min \{m \mid \mathcal{E}_1(n, m) \geq \beta\}, & \text{if } \mathcal{E}_1(n, n) \geq \beta, \\ \infty, & \text{if } \mathcal{E}_1(n, n) < \beta. \end{cases}$$

Thus, when n voters each choose randomly, the probability that any option meets or exceeds the threshold $t(n, \beta)$ is at most $\alpha = 1 - \beta$.

Finally, we define v , the number of extra votes needed, as

$$v(p^*, \beta) := \min \{n \mid \mathcal{E}_2(p^*, n, t(n, \beta)) \geq 1 - \beta\}$$

If workers have a bias of at least p^* toward a “popular” option (the remaining options being equiprobable), then when we ask $v(p^*, \beta)$ voters, the number of votes cast for the popular option passes the threshold (and all other options are below threshold) with probability at least β .

5.2. Quality control algorithm

AUTOMAN’s quality control algorithm, which gathers responses until it can choose the most popular answer not likely to be the product of random chance, proceeds as follows:

1. Set $b = \min \{m \mid t(m, \beta) \neq \infty\}$. Set $n = 0$.
2. Ask b workers to vote on a question with k options. Set $n = b + n$.
3. If any option has more than $t(n, \beta)$ votes, return the most frequent option as the answer.
4. Let $b = v(p^*, \beta)$ and repeat from step 2.

Figure 2 uses t to compute the smallest *fraction* of workers that need to agree for $\beta = 0.95$. As the number of tasks and the number of options increase, the proportion of workers needed to agree decreases. For example, for a 4-option question with 25 worker responses, only 48% (12 of 25) of workers must agree to meet the confidence threshold. This figure clearly demonstrates that quality control based on majority vote is neither necessary nor sufficient to limit outcomes based on random chance.

5.3. Multiple comparisons problem

Note that AUTOMAN must correct for a subtle bias that is introduced as the number of rounds—and correspondingly, the number of statistical tests—increases. This bias is called the *multiple comparisons problem*. As the number of hypotheses grows with respect to a fixed sample size, the probability that at least one true hypothesis will be incorrectly falsified by chance increases. Without the correction, AUTOMAN is susceptible to accepting low-confidence answers when the proportion of good workers is low. AUTOMAN applies a Bonferroni correction to its statistical threshold, which ensures that the *familywise error rate* remains at or below the $1 - \beta$ threshold set by the

programmer.¹⁰ We empirically evaluate the cost and time overhead for this correction in Section 7.4.

5.4. Quality control discussion

For AUTOMAN’s quality control algorithm to work, two assumptions must hold: (1) workers must be independent, and (2) random choice is the worst-case behavior for workers; that is, they will not deliberately pick the wrong answer. Workers may break the assumption of independence by masquerading as multiple workers, performing multiple tasks, or by colluding on tasks. We address each scenario below.

Scenario 1: Sybil Attack. A single user who creates multiple electronic identities for the purpose of thwarting identity-based security policy is known in the literature as a “Sybil attack.”⁶ The practicality of a Sybil attack depends directly on the feasibility of generating multiple identities.

Carrying out a Sybil attack on MTurk would be burdensome. Since MTurk provides a payment mechanism for workers, Amazon requires that workers provide uniquely identifying financial information, typically a credit card or bank account. These credentials are difficult to forge.

Scenario 2: One Worker, Multiple Responses. In order increase the pay or allotted time for a task, MTurk requires requesters to post a new HIT. This means that a single AUTOMAN task can span multiple MTurk HITs. MTurk provides a mechanism to ensure worker uniqueness for a single HIT that has multiple assignments, but it lacks the functionality to ensure that worker uniqueness is maintained across multiple HITs. For AUTOMAN’s quality control algorithm to be effective, AUTOMAN must be certain that workers who previously supplied responses cannot supply new responses for the same task.

Our workaround for this shortcoming is to use MTurk’s “qualification” feature inversely: once a worker completes a HIT, AUTOMAN grants the worker a special “disqualification” that precludes them from supplying future responses.

Scenario 3: Worker Collusion. While it is appealing to lower the risk of worker collusion by ensuring that workers are geographically separate (e.g., by using IP geolocation), eliminating this scenario entirely is not practical. Workers can collude via external channels (e-mail, phone, word-of-mouth) to thwart our assumption of independence. Instead, we opt to make the effort of thwarting defenses undesirable given the payout.

By spawning large numbers of tasks, AUTOMAN makes it difficult for any group of workers to monopolize them. Since MTurk hides the true number of assignments for a HIT, workers cannot know how many wrong answers are needed to defeat AUTOMAN’s quality control algorithm. This makes collusion infeasible. The bigger threat comes from workers who do as little work as possible to get compensated: previous research on MTurk suggests that random-answer spammers are the primary threat.²⁰

Random as worst case AUTOMAN’s quality control algorithm is based on excluding random responses. AUTOMAN gathers consensus not just until a popular answer is revealed, but also until its popularity is unlikely to be the product of random chance. As long as there is a crowd bias toward

the correct answer, AUTOMAN's algorithm will eventually choose it. Nevertheless, it is possible that workers could act maliciously and deliberately choose incorrect answers.

Random choice is a more realistic worst-case scenario: participants have an economic incentive not to deliberately choose incorrect answers. First, a correct response to a given task yields an immediate monetary reward. If workers know the correct answer, it is against their own economic self-interest to choose otherwise. Second, supposing that a participant chooses to forego immediate economic reward by deliberately responding incorrectly (e.g., out of malice), there are long-term consequences. MTurk maintains an overall ratio of accepted responses to total responses submitted (a "reputation" score), and many requesters only assign work to workers with high ratios (typically around 90%). Since workers cannot easily discard their identities for new ones, incorrect answers have a lasting negative impact on workers. We found that many MTurk workers scrupulously maintain their reputations, sending us e-mails justifying their answers or apologizing for having misunderstood the question.

6. SYSTEM ARCHITECTURE

AUTOMAN is implemented in tiers in order to cleanly separate three concerns: delivering reliable data to the end-user, interfacing with an arbitrary crowdsourcing system, and specifying validation strategies in a crowdsourcing system-agnostic manner. The programmer's interface to AUTOMAN is a domain-specific language embedded in the Scala programming language. The choice of Scala is to maintain full interoperability with existing Java Virtual Machine code. The DSL abstracts questions at a high level as question functions.

Upon executing a question function, AUTOMAN computes the number of tasks to schedule, the reward, and the timeout; marshals the question to the appropriate backend; and returns immediately, encapsulating work in a Scala Future. The runtime memoizes all responses in case the user's program crashes. Once quality control goals are satisfied, AUTOMAN selects and returns an answer.

Each tier in AUTOMAN is abstract and extensible. The default quality control strategy implements the algorithm described in Section 5.2. Programmers can replace the default strategy by implementing the `ValidationStrategy` interface. The default backend is MTurk, but this backend can be replaced with few changes to client code by supplying an `AutomanAdapter` for a different crowdsourcing platform.

7. EVALUATION

We implemented three sample applications using AUTOMAN: a semantic image-classification task using checkboxes (Section 7.1), an image-counting task using radio buttons (Section 7.2), and an optical character recognition (OCR) pipeline using text entry (Section 7.3). These applications were chosen to be representative of the kinds of problems that remain difficult even for state-of-the-art algorithms. We also performed a simulation using real and synthetic traces to explore AUTOMAN's performance as confidence and worker quality is varied (Section 7.4).

7.1. Which item does not belong?

Our first sample application asks users to identify which object does not belong in a collection of items. This kind of task requires both image- and semantic-classification capability, and is a component in clustering and automated construction of ontologies. Because tuning of AUTOMAN's parameters is unnecessary, relatively little code is required to implement this program (27 lines in total).

We gathered 93 responses from workers during our sampling runs. Runtimes for this program were on the order of minutes, but there is substantial variation in runtime given the time of the day. Demographic studies of MTurk have shown that the majority of workers on MTurk are located in the United States and in India.¹¹ These findings largely agree with our experience, as we found that this program (and variants) took upward of several hours during the late evening hours in the United States.

7.2. How many items are in this picture?

Counting the number of items in an image also remains difficult for state-of-the-art machine learning algorithms. Machine-learning algorithms must integrate a variety of feature detection and contextual reasoning algorithms in order to achieve a fraction of the accuracy of human classifiers.¹⁸ Moreover, vision algorithms that work well for all objects remain elusive.

Counting tasks are trivial with AUTOMAN. We created an image processing pipeline that takes a search string as input, downloads images using Google Image Search, resizes the images, uploads the images to Amazon S3, obscures the URLs using TinyURL, and then posts the question "How many `$items` are in this image?"

We ran this task eight times, spawning 71 question instances at the same time of the day (10 a.m. EST), and employing 861 workers. AUTOMAN ensured that for each of the 71 questions asked, no worker was could participate more than once. Overall, the typical task latency was short. We found that the mean runtime was 8 min, 20 s and that the median runtime was 2 min, 35 s.

The mean is skewed upward by the presence of one long-running task that asked "How many spoiled apples are in this image?" The difference of opinion caused by the ambiguity of the word "spoiled" caused worker answers to be nearly evenly distributed between two answers. This ambiguity forced AUTOMAN to collect a large number of responses in order to meet the desired confidence level. AUTOMAN handled this unexpected behavior correctly, running until statistical confidence was reached.

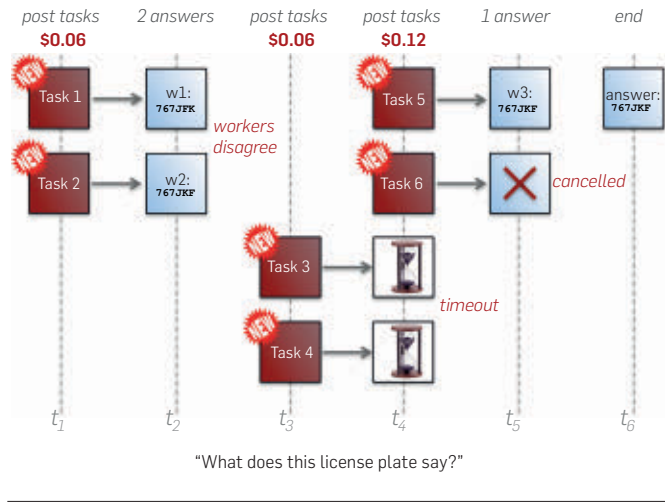
7.3. Automatic license plate recognition

Our last application is the motivating example shown in Figure 1, a program that performs automatic license plate recognition (ALPR). Although ALPR is now widely deployed using distributed networks of traffic cameras, it is still considered a difficult research problem,⁸ and academic literature on this subject spans nearly three decades.⁵ While state-of-the-art systems can achieve accuracy near 90% under ideal conditions,

^b The MediaLab LPR database is available at <http://www.medialab.ntua.gr/research/LPRdatabase.html>.

these systems require substantial engineering in practice.⁴ False positives have dramatic negative consequences in unsupervised ALPR systems as tickets are issued to motorists automatically. A natural consequence is that even good unsupervised image-recognition algorithms may need humans in the loop to audit results and to limit false positives.

Figure 3. A sample trace from the ALPR application shown in Figure 1. AutoMAN correctly selects the answer 767JKF, spending a total of \$0.18. Incorrect, timed-out, and cancelled tasks are not paid for, saving programmers money.



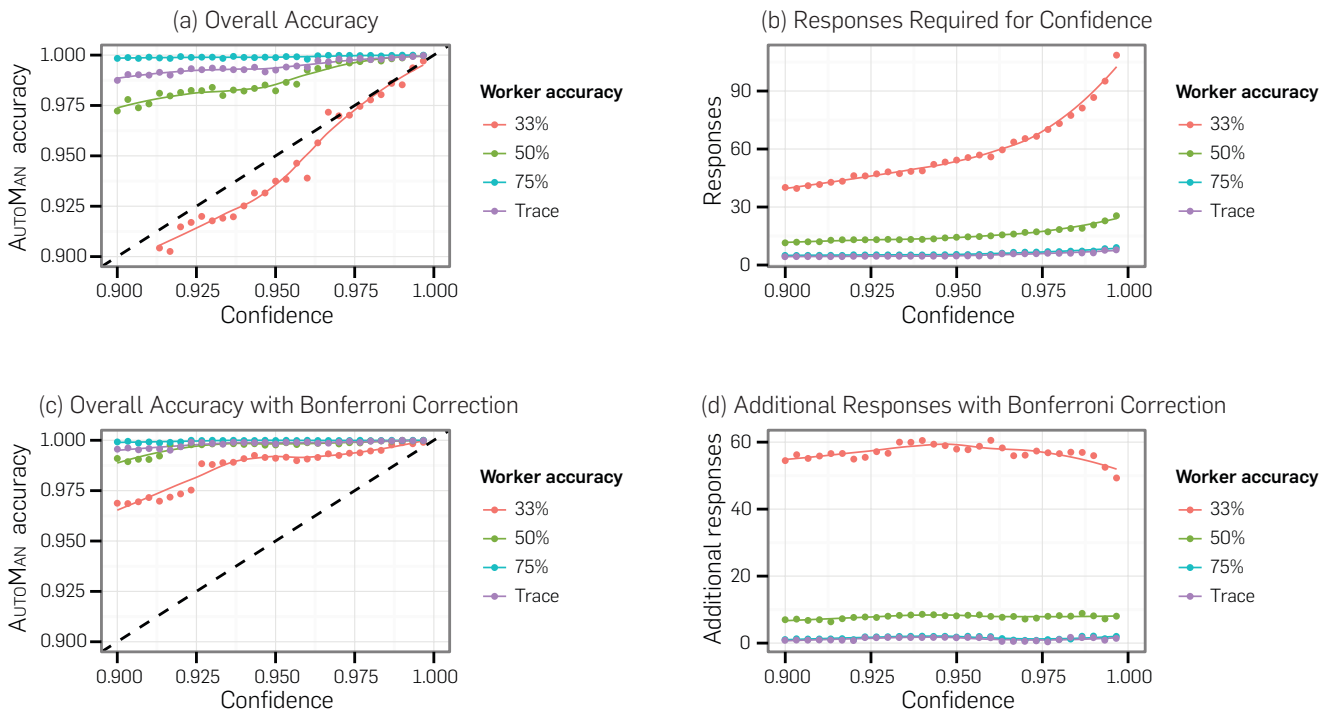
Using AUTOMAN to engage humans to perform this task required only a few hours of programming time and AUTOMAN's quality control ensures that it delivers results that match or exceed the state-of-the-art on even the most difficult cases. We evaluated the ALPR application using the MediaLab LPR^b database. Figure 3 shows a sample trace for a real execution.

The benchmark was run twice on 72 of the “extremely difficult” images, for a total of 144 license plate identifications. Overall accuracy was 91.6% for the “extremely difficult” subset. Each task cost an average of 12.08 cents, with a median latency of less than 2 min per image. AUTOMAN runs all identification tasks in parallel: one complete run took less than 3 h, while the other took less than 1 h. These translate to throughputs of 24 and 69 plates/h. While the AUTOMAN application is slower than computer vision approaches, it is simple to implement, and it could be used for only the most difficult images to increase accuracy at low cost.

7.4. Simulation

We simulate AUTOMAN's ability to meet specified confidence thresholds by varying two parameters, the minimum confidence threshold β , where $0 < \beta < 1$ (we used 50 levels of β), and the probability that a random worker chooses the correct answer $p_r \in \{0.75, 0.50, 0.33\}$. We also simulate worker responses drawn from trace data (“trace”) for the “Which item does not belong?” task (Section 7.1). For each setting of

Figure 4. These plots show the effect of worker accuracy on (a) overall accuracy and (b) the number of responses required on a five-option question. “Trace” is a simulation based on real response data while the other simulations model worker accuracies of 33%, 50%, and 75%. Each round of responses ends with a hypothesis test to decide whether to gather more responses, and AutoMAN must schedule more rounds to reach the confidence threshold when worker accuracy is low. Naively performing multiple tests creates a risk of accepting a wrong answer, but the Bonferroni correction eliminates this risk by increasing the confidence threshold with each test. Using the correction, AutoMAN (c) meets quality control guarantees and (d) requires few additional responses for real workers.



β and p_r , we run 10,000 simulations and observe AUTOMAN's response. We classify responses as either *correct* or *incorrect* given the ground truth. *Accuracy* is the mean proportion of correct responses for a given confidence threshold. *Responses required* is the mean number of workers needed to satisfy a given confidence threshold.

Figure 4a and 4b shows accuracy and the number of required responses as a function of β and p_r , respectively. Since the risk of choosing a wrong answer increases as the number of hypothesis tests increases (the “multiple comparisons” problem), we also include figures that show the result of correcting for this effect. Figure 4c shows the accuracy and Figure 4d shows the increase in the number of responses when we apply the Bonferroni bias correction.¹⁰

These results show that AUTOMAN's quality control algorithm is effective even under pessimistic assumptions about worker quality. AUTOMAN is able to maintain high accuracy in all cases. Applying bias correction ensures that answers meet the programmer's quality threshold even when worker quality is low. This correction can significantly increase the number of additional worker responses required when bad workers dominate. However, worker accuracy tends to be closer to 60%,²⁰ so the real cost of this correction is low.

8. RELATED WORK

Programming the Crowd. While there has been substantial *ad hoc* use of crowdsourcing platforms, there have been few efforts to manage workers programmatically beyond MTurk's low-level API.

TurKit Script extends JavaScript with a templating feature for common MTurk tasks and adds checkpointing to avoid re-submitting tasks if a script fails.¹⁵ CrowdForge and JabberWocky wrap a MapReduce-like abstraction on MTurk tasks.^{1, 13} Unlike AUTOMAN, neither TurKit nor CrowdForge automatically manage scheduling, pricing, or quality control; JabberWocky uses fixed pricing along with a majority-vote based quality-control scheme.

CrowdDB models crowdsourcing as an extension to SQL for crowdsourcing database cleansing tasks.⁹ The query planner tries to minimize the expense of human operations. CrowdDB is not general-purpose and relies on majority voting as its sole quality control mechanism.

Turkomatic crowdsources an entire computation, including the “programming” of the task itself.¹⁴ Turkomatic can be used to construct arbitrarily complex computations, but Turkomatic does not automatically handle budgeting or quality control, and programs cannot be integrated with a conventional programming language.

Quality Control. CrowdFlower is a commercial web service.¹⁷ To enhance quality, CrowdFlower seeds questions with known answers into the task pipeline. CrowdFlower incorporates methods to programmatically generate these “gold” questions to ease the burden on the requester. This approach focuses on establishing trust in particular workers.¹² By contrast, AUTOMAN does not try to estimate worker quality, instead focusing on worker agreement.

Shepherd provides a feedback loop between task requesters and task workers in an effort to increase quality; the idea is

to *train* workers to do a particular job well.⁷ AUTOMAN requires no feedback between requester and workers.

Soylent crowdsources finding errors, fixing errors, and verifying the fixes.³ Soylent can handle open-ended questions that AUTOMAN currently does not support. Nonetheless, unlike AUTOMAN, Soylent's approach does not provide any quantitative quality guarantees.

9. CONCLUSION

Humans can perform many tasks with ease that remain difficult or impossible for computers. We present AUTOMAN, the first *crowdprogramming* system. Crowdprogramming integrates human-based and digital computation. By automatically managing quality control, scheduling, and budgeting, AUTOMAN allows programmers to easily harness human-based computation for their applications.

AUTOMAN is available at www.automan-lang.org.

Acknowledgments

This work was supported by the National Science Foundation Grant No. CCF-1144520 and DARPA Award N10AP2026. Andrew McGregor is supported by the National Science Foundation Grant No. CCF-0953754. The authors gratefully acknowledge Mark Corner for his support and initial prompting to explore crowdsourcing. 

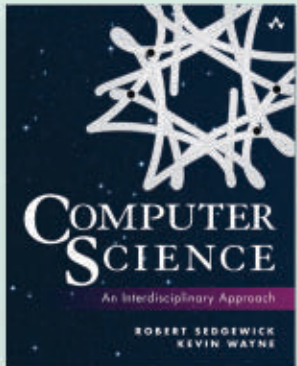
References

- Ahmad, S., Battle, A., Malkani, Z., Kamvar, S. The Jabberwocky programming environment for structured social computing. In *UIST 2011*, 53–64.
- Barowy, D.W., Curtsinger, C., Berger, E.D., McGregor, A. AutoMan: A platform for integrating human-based and digital computation. In *OOPSLA 2012*, 639–654.
- Bernstein, M.S., Little, G., Miller, R.C., Hartmann, B., Ackerman, M.S., Karger, D.R., Crowell, D., Panovich, K. Soylent: A word processor with a crowd inside. In *UIST 2010*, 313–322.
- Chang, G.-L., Zou, N. *ITS Applications in Work Zones to Improve Traffic Operation and Performance Measurements*. Technical Report MD-09-SP708B4G, Maryland Department of Transportation State Highway Administration, May.
- Davies, P., Emmott, N., Ayland, N. License plate recognition technology for toll violation enforcement. In *IEEE Colloquium on Image Analysis for Transport Applications* (Feb 1990), 7/1–7/5.
- Douceur, J.R. The Sybil attack. In *IPTPS 2001*, 251–260.
- Dow, S., Kulkarni, A., Bunge, B., Nguyen, T., Klemmer, S., Hartmann, B. Shepherding the crowd: Managing and providing feedback to crowd workers. In *CHI 2011*, 1669–1674.
- Due, S., Ibrahim, M., Shehata, M., Badawy, W. Automatic license plate recognition (ALPR): A state of the art review. *IEEE Trans. Circ. Syst. Video Technol.* 23 (2012), 311–325.
- Franklin, M.J., Kossmann, D., Kraska, T., Ramesh, S., Xin, R. CrowdDB: Answering queries with crowdsourcing. In *SIGMOD 2011*, 61–72.
- Holm, S. A simple sequentially rejective multiple test procedure. *Scand. J. Stat.* 6, 2 (1979), 65–70.
- Ipeirotis, P.G. *Demographics of Mechanical Turk*. Technical Report CeDER-10-01, NYU Center for Digital Economy Research, 2010.
- Ipeirotis, P.G., Provost, F., Wang, J. Quality management on Amazon mechanical Turk. In *HCOMP 2010*, 64–67.
- Kittur, A., Smus, B., Khamkar, S., Kraut, R.E. CrowdForge: Crowdsourcing Complex Work.
- Kulkarni, A.P., Can, M., Hartmann, B. Turkomatic: Automatic recursive task and workflow design for mechanical Turk. In *CHI 2011*, 2053–2058.
- Little, G., Chilton, L.B., Goldman, M., Miller, R.C. TurKit: Human computation algorithms on mechanical Turk. In *UIST 2010*, 57–66.
- Marge, M., Banerjee, S., Rudnick, A. Using the Amazon mechanical Turk for transcription of spoken language. In *ICASSP 2010*, 5270–5273, Mar.
- Oleson, D., Hester, V., Sorokin, A., Laughlin, G., Le, J., Biewald, L. Programmatic gold: Targeted and scalable quality assurance in crowdsourcing. In *HCOMP 2011*, 43–48.
- Parikh, D., Zitnick, L. Human-debugging of machines. In *NIPS CSS 2011*.
- Shahaf, D., Amir, E. Towards a theory of AI completeness. In *Commonsense 2007*.
- Tamir, D., Kanth, P., Ipeirotis, P. Mechanical Turk: Now with 40.92% spam, Dec 2010. www.behind-the-enemy-lines.com.

Daniel W. Barowy, Charlie Curtsinger, Emery D. Berger, and Andrew McGregor ([dbarowy, charlie, emery, mcgregor]@cs.umass.edu), College of

Information and Computer Sciences, University of Massachusetts Amherst, 140 Governors Drive, Amherst, MA.



ISBN-13: 978-0-13-449383-1

A CS textbook for the 21st century

- Programming with a purpose
- Scientific context
- Theory of computing
- von Neumann machines
- Circuits and CPUs

Computer Science

An Interdisciplinary Approach

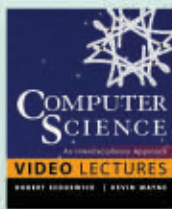
Robert Sedgewick & Kevin Wayne
Princeton University

1150 pages • 450 figures •
150 programs • 650 exercises

Available at informit.com/sedgewick

Additional Resources for Teachers and Learners

Studio-produced
video lectures



ISBN-13: 978-0-13-449383-1

Free online content and
resources available at:
<http://introcs.cs.princeton.edu/java>



Pearson

[CONTINUED FROM P. 112] it; they just announced it in the Federal Register as a proposed standard. We quickly realized the key size was too small and needed to be enlarged.

DIFFIE: I had an estimate roughly of half a billion dollars to break it. We eventually decided it could be done for \$20-ish million.

HELLMAN: And because of Moore's Law, it would only get cheaper.

DIFFIE: If you can make a cryptographic system that's good, it's usually not hard to make one that's effectively unbreakable. So it takes some explaining if you make the key size small enough that somebody might conceivably search through it.

HELLMAN: So in March 1975, NBS announced the DES and solicited comments and criticism. And we were naive enough to think they actually were open to improving the standard. Five months later, it was clear to us the key size problem was intentional and the NSA was behind it. If we wanted to improve DES—and we did—we had a political fight on our hands.

That fight was partly about your work on public key cryptography.

MARTIN: There was a lot that led up to that idea. The DES announcement

"If you can make a cryptographic system that's good, it's usually not hard to make one that's effectively unbreakable."

suggested the value of trap door ciphers. It became clear to us that NSA wanted secure encryption for U.S. communications, but still wanted access to foreign ones. Even better than DES' small key size would be to build in a trap door that made the system breakable by NSA—which knows the trap door information—but not by other nations. It's a small step from there to public key cryptography, but it still took us time to see.

Whit, you have also said you were inspired by John McCarthy's paper about buying and selling through so-called "home information terminals."



Whitfield Diffie

DIFFIE: I was concerned with two problems and didn't realize how closely related they were. First, I had been thinking about secure telephone calls since 1965, when a friend told me—mistakenly, as it turned out—that the NSA encrypted the telephone traffic within its own building. From my countercultural point of view, though, my understanding of a secure telephone call was: I call you, and nobody else in the world can understand what we're talking about. I began thinking about what we call the key-management problem.

In 1970, about the time I got to Stanford, John McCarthy presented the paper that you note. I began to think about electronic offices and what you would do about a signature, because signatures on paper depend so heavily on the fact that they're hard to copy, and digital documents can be copied exactly.

So in the spring of 1975, as you were preparing your critique of DES, you came to the solution to both problems.

DIFFIE: I was living at John McCarthy's house, and I was trying to combine what is called identification, friend or foe (IFF), which is a process by which a Fire Control radar challenges an aircraft and expects a correctly encrypted response, and what is called one-way enciphering, which is used in UNIX to avoid having the password table be secret. One of these protects you from the compromise of the password table, and the other protects you from someone eavesdropping on the transmission of your password.

You came to the concept of what we now call digital signatures, constructions in which somebody can judge the correctness of the signature but cannot have generated it.

DIFFIE: Only one person can generate it, but anybody can judge its correctness. And then a few days later, I realized this could be used to solve the problem I'd been thinking of since 1965. At that point, I realized I really had something. I told Mary about it as I fed her dinner and then went down the hill to explain it to Marty.

HELLMAN So then we had the problem of coming up with a system that would actually implement it practical-



Martin E. Hellman

ly, and some time later we met Ralph Merkle, who had come up with related but slightly different ideas at Berkeley as a master's student. The algorithm I came up with was a public key distribution system, a concept developed by Merkle. Whit and I didn't put names on the algorithm, but I've argued it should be called Diffie-Hellman-Merkle, rather than the Diffie-Hellman Key Exchange, as it now is.

The NSA was not happy you intended to publish your results.

HELLMAN: NSA was very upset at our publishing in an area where they thought they had a monopoly and could control what was published.

Marty, you have been at Stanford ever since. Whit, you left Stanford in 1978 to work at Bell Northern Research, and later went to Sun Microsystems. And you now are working on a project to document the history of cryptography.

DIFFIE: There have been some major shifts in cryptographic technology in the latter half of the 20th century; public key is only one of them. I am trying to write the history of some others before all the people who worked on them die off.

Marty, you're writing a book about your marriage and nuclear weapons.

HELLMAN: Starting about 35 years ago, my interests shifted from cryptography to the bigger problems in the world, particularly nuclear weapons and how fallible human beings are going to survive having that kind of power. What got me started was wanting to save my marriage, which at that time was in trouble. Dorothea and I not only saved our marriage, but recaptured the deep love we felt when we first met. The changes needed to transform our marriage are the same ones needed to build a more peaceful, sustainable world. But it has kind of come full circle, because as we become more and more wired, cyber insecurity may become an existential threat. The global part of our effort is really about solving the existential threats created by the chasm between the God-like physical power technology has given us and our maturity level as a species, which is at best that of an irresponsible adolescent. 

Leah Hoffmann is a technology writer based in Piermont, NY.

© 2016 ACM 0001-0782/16/06 \$15.00

Q&A

Finding New Directions in Cryptography

Whitfield Diffie and Martin Hellman on their meeting, their research, and the results that billions use every day.

LIKE MANY DEVELOPMENTS WE NOW take for granted in the history of the Internet, public key cryptography—which provides the ability to communicate securely over an insecure channel—followed an indirect path into the world. When ACM A.M. Turing Award recipients Martin Hellman and Whitfield Diffie began their research, colleagues warned against pursuing cryptography, a field then dominated by the U.S. government. Their 1976 paper “New Directions in Cryptography” not only blazed a trail for other academic researchers, but introduced the ideas of public-key distribution and digital signatures.

How did you meet?

DIFFIE: In the summer of 1974, my wife and I traveled to Yorktown Heights (NY) to visit a friend who worked for Alan Konheim at IBM.

You’re talking about the head of the IBM mathematics group and author of *Cryptography: A Primer*, who subsequently moved to the University of California, Santa Barbara.

DIFFIE: Konheim said he couldn’t tell me very much because of a secrecy order, but he did mention that his friend Marty Hellman had been there a few months ago. He said, “I couldn’t tell him anything either, but you should look him up when you get back to Stanford, because two people can work on a problem better than one.”

HELLMAN: So Whit gives me a call. Whit, you were up in Berkeley at the time?



DIFFIE: I was staying with Leslie Lamport.

HELLMAN: I think I set up a half-hour meeting in my office, which went on for probably two hours, and at the end of it, I said, “Look, I’ve got to go home to watch my daughters, but can we continue this there?” Whit came to our house and we invited him and his wife, Mary, to stay for dinner, and as I remember we ended the conversation around 11 o’clock at night.

The two of you worked together for the next four years.

HELLMAN: Whit had been traveling around the country and I tried to fig-

ure out ways to keep him at Stanford. I found a small amount of money in a research grant that I could use. A lot of good things came of that.

Among them was a vigorous critique of the Data Encryption Standard (DES), a symmetric-key algorithm developed at IBM.

HELLMAN: DES came full-blown from the brow of Zeus. “Zeus,” in this case, was NBS, the National Bureau of Standards, or NSA, the National Security Agency, or IBM, or some combination. They didn’t tell us how they had come up with [CONTINUED ON P. 110]



Association for
Computing Machinery



ACM Seeks New Editor(s)-in-Chief for *ACM Interactions*

The ACM Publications Board is seeking a volunteer editor-in-chief or co-editors-in-chief for its bimonthly magazine *ACM Interactions*.

ACM Interactions is a publication of great influence in the fields that envelop the study of people and computers. Every issue presents an array of thought-provoking commentaries from luminaries in the field together with a diverse collection of articles that examine current research and practices under the HCI umbrella.

For more about *ACM Interactions*, see <http://interactions.acm.org>

Job Description The editor-in-chief is responsible for organizing all editorial content for every issue. These responsibilities include: proposing articles to perspective authors; overseeing the magazine's editorial board and contributors; creating new editorial features, columns and much more.

An annual stipend will be available for the hiring of an editorial assistant.

Financial support will also be provided for any travel expenses related to this position.

Eligibility Requirements The EiC search is open to applicants worldwide. Experience in and knowledge about the issues, challenges, and advances in human-computer interaction is a must.

The ACM Publications Board has convened a special search committee to review all candidates for this position.

Please send your CV and vision statement of 1,000 words or less expressing the reasons for your interest in the position and your goals for *Interactions* to the search committee at eicsearch@interactions.acm.org, Subject line: RE: Interactions.

The deadline for submissions is June 1, 2016 or until position is filled.

The editorship will commence on September 1, 2016.

You must be willing and able to make a three-year commitment to this post.



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」



Panelists:

Otmar Hilliges
(ETH Zürich)

Hrvoje Benko
(Microsoft Research)

Alex Olwal
(Google Inc.)

Moderator:

Aitor Rovira
(Nara Institute of
Science and Technology)